Advanced Composites Manufacturing: Process Modeling/Analysis of Complex Engineering Structures Manufactured by Resin Transfer Molding

R. Kanapady[1] K. K. Tamma[2] ,

Department of Mechanical Engineering
111 Church Street S.E.
University of Minnesota
Minneapolis, MN 55455, USA

[1]Doctoral research student
[2]Professor, To receive correspondence

SF 298 MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1 January 1999	Final Report

4. TITLE AND SUBTITLE
Advanced Composites Manufacturing:Process Modeling/Analysis of Complex Engineering Structures Manufactured by RTM

5. FUNDING NUMBERS
DAAH04-96-1-0172

6. AUTHOR(S)
R. Kanapady and K. K. Tamma

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
Department of Mechanical Engineering
University of Minnesota
111 Church Street S.E.
Minneapolis, MN 55455

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 35543.2-MA

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The present work employs some of the major elements of the new computers which are promising parallel computer architectures of the future, and describes some recent developments in the finite element analysis of process modeling and manufacturing applications of composites that are designed to exploit these major elements for large scale engineering applications. The work includes finite element computational schemes, data structures and interprocessor communication strategies for the implementation of large scale practical advanced manufacturing simulations with particular emphasis on isothermal resin transfer molding (RTM) process manufacturing simulations on the symmetric multiprocessor. For process modeling studies thin shell composite mold configurations are used to illustrate the validity of the present implementation of: i) the traditional explicit control volume-finite element, and (ii) a recently developed and new pure finite element implicit methodology in conjunction with a diagonal preconditioned conjugate gradient solver for parallel computations on the SGI Power Challenge and the SGI Origin2000 symmetric multiprocessor machines. The techniques developed are applied to large scale problems to demonstrate the practical applicability to realistic industrial size problems.

14. SUBJECT TERMS	15. NUMBER IF PAGES
Resin Transfer Molding, Process Modeling, Computational Simulations, Finite Elements, High Performance Computing	55
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Abstract

Parallel computing is fast becoming an inexpensive alternative to the standard super-computing approach for solving very large scale problems that arise in scientific and engineering applications. Science and engineering applications involved in product design and analysis are subjected to competitive and regulatory constraints such as the need to shorten design cycles, reduce product cost, meet increasingly stringent government regulations, improve quality and safety, and reduce environmental impact. These constraints have increased the need for accurate product and component simulation and demand analysts for simulations of unprecedented scale and complexity making it impossible to perform simulation in a sequential fashion using computers of conventional architecture. With single RISC-based microprocessor performance improve every year, computer hardware industry is providing high-performance parallel architectures which are built on commodity parts and compatible with workstation systems. One such computer architecture is symmetric multiprocessors (SMP). Furthermore, PC's to large computer systems are rapidly becoming symmetric multiprocessor machines. These architectures offer the level of computational performance and increased memory capacity needed for large-scale computing. Because the underlying architecture is different than traditional uniprocessor or vector supercomputers, this increased performance is most fully available to programs designed to take advantage of the new architecture design.

The present work employs some of the major elements of these new computers which are promising parallel computer architectures of the future, and describes some recent developments in the finite element analysis of process modeling and manufacturing applications of composites that are designed to exploit these major elements for large scale engineering applications. The work includes finite element computational schemes, data structures and interprocessor communication strategies for the implementation of large scale practical advanced manufacturing simulations with particular emphasis on isothermal resin transfer molding (RTM) process manufacturing simulations on the symmetric multiprocessor. For process modeling studies thin shell composite mold configurations are used to illustrate the validity of the present implementation of: i) the traditional explicit control volume-finite element, and (ii) a recently developed and new pure finite element implicit methodology in conjunction with a diagonal preconditioned conjugate gradient solver for parallel computations on the SGI Power Challenge and the SGI Origin2000 symmetric multiprocessor machines. The techniques developed are applied to large scale problems to demonstrate the practical applicability to realistic industrial size problems.

The results indicate that the current generation SMP's with medium to large number of processors can be effectively used as a supercomputer (Massively Parallel Platforms) for large scale complex geometries in scientific and engineering problems. The recently developed and new pure finite element implicit methodology is shown to be physically accurate and computationally superior compared to the explicit control-volume finite element algorithm (explicit CV-FE) on different SMP platforms (SGI Power Challenge and SGI Origin2000). For large scale problems, explicit CV-FE based modeling/analysis become impossible to analyze within *Reasonable Time* or realistically impossible even with parallel processing for large scale problems. In view of such considerations, as an illustration, a large complex finite element mesh of 809,505 elements and 405,327 nodes has been successfully analyzed using the pure finite element implicit methodology and 40 processors within a *Reasonable Time* of 4.72 hrs. The proposed parallel approaches have excellent parallel efficiency and scalability when compared to other relevant results published in the literature with the highest degree of portability of the software code to wide range of parallel architectures.

Contents

List of Figures

List of Tables

Composite Process Modeling Applications

1.1 Introduction

In this research the implementation of a finite element-control volume explicit mold filling technique that is traditionally advocated and a pure finite element implicit mold filling technique for isothermal resin transfer molding process technology for composites manufacturing is described. The computational complexity of these applications are, at a minimum, similar or comparably more than other finite element applications such as non-linear static structural analysis, dynamic structural analysis or fluid flow analysis. In fact the computational complexity of mold filling finite element algorithms is somewhat more complicated as it involves fluid flow front tracking computations in addition to time stepping and flow calculations. Because of these reasons, this application is selected for parallel implementation studies.

1.2 Resin transfer mold filling

Computational developments in manufacturing simulations are becoming important in process modeling and optimization of composite manufacturing process such as Resin Transfer Molding (RTM). Composite structures made of these materials are increasingly being used in the military, automotive, aerospace and electronic industries due to their excellent properties such as the high stiffness to weight ratio, long fatigue life, increased corrosion resistance and ability to manufacture complicated consolidated parts. Resin transfer mold filling process involves the injection of the resin matrix into a mold cavity filled with fiber mat. After the resin cures, the finished part is removed from the mold. The process is schematically illustrated in Fig 1.1. The Resin Transfer Molding is also emerging as a technology for manufacturing large components of fiber reinforced composite structures. As the part of the Comanche helicopter structural component, a composite keel beam, Fig 1.2, shows such a complex geometry as an illustration.
Manufacturing of such large composite structures poses significant challenges and computational difficulties. The resin transfer molding process is in general a complex flow phenomenon involving a non-isothermal polymer resin flowing through a mold cavity filled with fiber preform and can involve multiple phase flow regions and are further complicated by the resin kinetic reactions during the thermal curing of the resin. For preliminary design studies, isothermal simulations provide a good design and is the focus of the present study. Thus a number of trial runs are involved in obtaining a successful part. It is almost impractical and/or ineffective to conduct large scale simulations on traditional computing platforms. These trials runs are expensive (as related to computational time and memory requirements) for large complex geometric parts, Fig. 2.16, which involves hundreds of thousands of degrees of freedom is typically illustrative application for conducting an analysis to accurately capture flow fronts inside the mold. Parallel computing is a viable alternative to reduce the computational times and to overcome the memory requirements for handling such problems. Literature shows that the finite element method forms the basis of the resin flow simulations inside

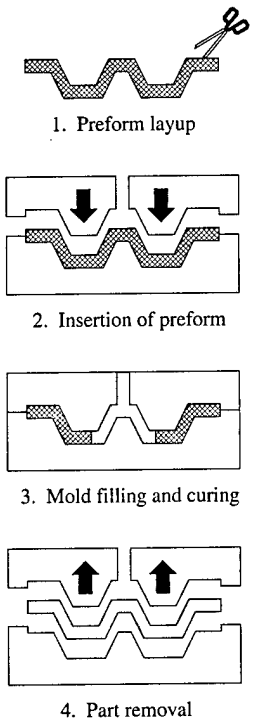


Figure 1.1: Resin transfer molding process

the mold cavity and have been mostly carried out by various researchers [35–39] for fairly small structural components on traditional computing platforms.

1.2.1 Isothermal filling

Governing Mold Filling Equations: Thin composite sections

For 3-D geometries governed by Darcy's law, the following assumptions are made:

1. The polymer resin is incompressible. The viscosity is taken to be constant during the whole process. The simulation assumes that the resin remains at a constant temperature and there is no significant difference in the resin temperature during the simulation, thereby representing isothermal filling conditions.
2. The flow is governed by Darcy's law, and the Reynolds number is small so that the inertia terms in the equation of motion can be neglected.

These assumptions agree with the earlier works [3, 2, 7] involving two-dimensional models.

It is assumed that the flow is pressure driven and the average velocity components are related to the pressure gradients by Darcy's law given as

$$\hat{\boldsymbol{u}} = -\frac{\boldsymbol{K}}{\mu}\nabla P \tag{1.1}$$

where ∇P represents the pressure gradient, and μ represents the effective viscosity. The flow satisfies the continuity equation

$$\nabla \,.\, \boldsymbol{u} = 0 \tag{1.2}$$

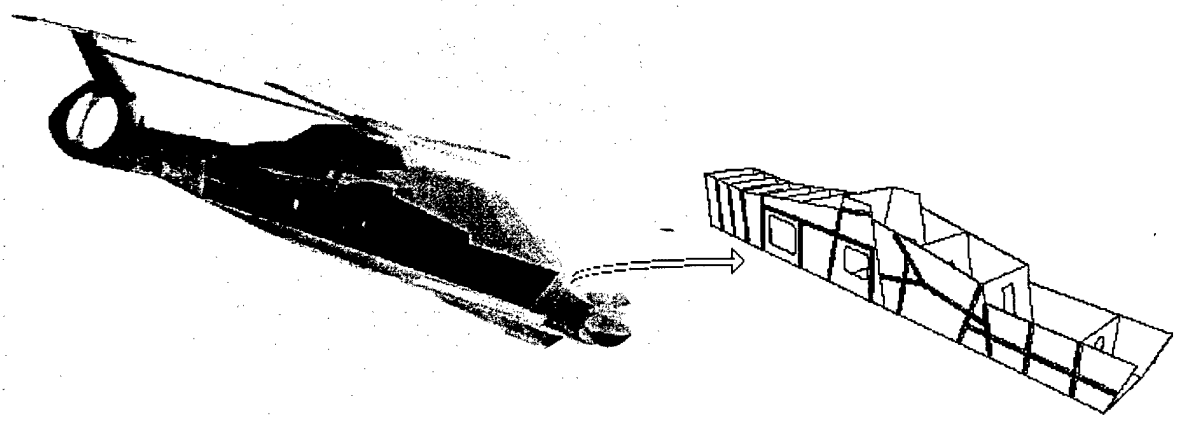


Figure 1.2: Schematic of Commanche keel beam

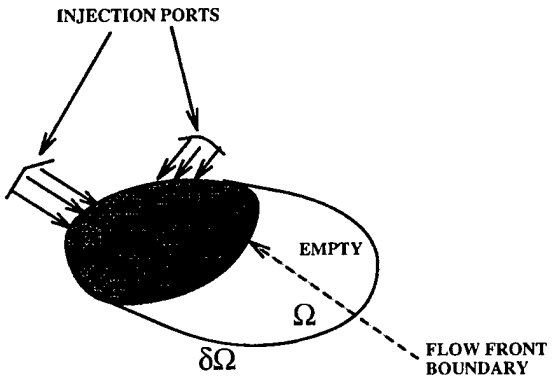


Figure 1.3: General partially filled mold cavity

Using Eq.(1.1) in the above continuity equation, an equation governing the pressure distribution in a complex 3-D configuration is obtained as

$$\nabla \cdot \left(\frac{\boldsymbol{K}}{\mu} \nabla P \right) = 0 \tag{1.3}$$

and the permeability matrix, which depends on the fiber arrangement, is defined as

$$[\ K\ ] = \begin{bmatrix} K_{xx} & K_{xy} & K_{xz} \\ K_{yx} & K_{yy} & K_{yz} \\ K_{zx} & K_{zy} & K_{zz} \end{bmatrix} \tag{1.4}$$

By employing local element level permeabilities, the variation of the permeabilities throughout the mold geometry can be appropriately taken into account. For macroscopic mold filling simulations it is assumed that the permeability matrix is known *a priori*. The boundary conditions for the above modified Laplacian equation for the pressure are zero pressure at the flow front and zero pressure gradient in the direction normal to the mold wall at the mold surface since no leakage is assumed at the mold walls. The conditions at the injection ports can be either injection flow rate conditions or injection pressure conditions. In mathematical notation:

At mold surface: $\frac{\partial P}{\partial n} = 0$ on Γ_1
At resin front: $P = 0$ on Γ_2
At mold inlet: $P = P_0$ (prescribed pressure), or
$q = q_0$ (prescribed flow rate)

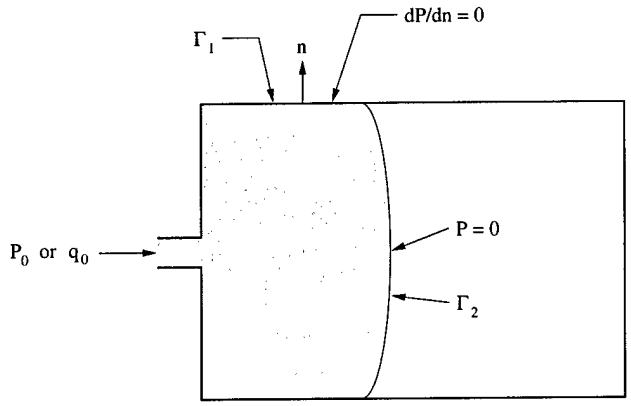


Figure 1.4: Schematic mold geometry and the applied boundary condition

Figure 1.4 shows a schematic of the mold geometry and the applied boundary conditions.

1.2.2 Traditional Explicit Filling: Control Volume-Finite Element (CV-FE) Approach

Various techniques exist to identify the filled regions which are determined based on the filled finite element nodes. In principle, all of these techniques involve updating the location of the flow front at each time step using the Darcy velocity field and the computed mass fluxes into the discretized mold regions.
This method involves associating a parameter called the fill factor with each node that is associated with a control volume region. The fill factor indicates whether a given control volume region is full or empty. In the explicit CV-FE approach, the transient mold filling problem is treated as a quasi-steady state problem and the time increment employed allows for only one region to be completely filled during a single time increment step. In certain conditions, several control volume regions may be filled simultaneously if their fill time steps are the same as the minimum time increment.

Space discretization

The governing equation for the pressure distribution is given as:

$$\nabla \cdot \left(\frac{\boldsymbol{K}}{\mu} \nabla P \right) = 0 \tag{1.5}$$

Invoking the traditional Galerkin weighted residual formulation, we have

$$\int_{\Omega} W \left[\nabla \cdot \left(\frac{\boldsymbol{K}}{\mu} \nabla P \right) \right] d\Omega = 0 \tag{1.6}$$

From the application of the Green-Gauss theorem, which states that,

$$\int_{\Omega} u \, (\nabla \cdot \boldsymbol{v}) \; d\Omega = \int_{\Gamma} u \, (\boldsymbol{v} \cdot \boldsymbol{n}) \; d\Gamma - \int_{\Omega} (\nabla u \cdot \boldsymbol{v}) \; d\Omega \tag{1.7}$$

the above Eq.(1.6) can be written as:

$$\int_{\Gamma} W \frac{\boldsymbol{K}}{\mu} \, (\nabla P \cdot \boldsymbol{n}) \; d\Gamma - \int_{\Omega} \nabla W \; \frac{\boldsymbol{K}}{\mu} \; \nabla P \; d\Omega = 0 \tag{1.8}$$

Letting

$$W = N_i$$
$$P = N_i\ P_i \tag{1.9}$$

where N_i are the finite element spatial shape functions and P_i are the nodal values of the pressure, Eq.(1.8) reduces to:

$$\left[\int_\Omega \nabla N_i\ \frac{\boldsymbol{K}}{\mu}\ \nabla N_j\ d\Omega\right] P_i = \int_\Gamma N_i \frac{\boldsymbol{K}}{\mu}\ (\nabla P \cdot \boldsymbol{n})\ d\Gamma \tag{1.10}$$

This discretized system of equations can be written as:

$$\boldsymbol{K_s}\ \boldsymbol{P}\ =\ \boldsymbol{q} \tag{1.11}$$

where

$$\boldsymbol{K_s} = \int_\Omega \boldsymbol{B}^T \frac{\boldsymbol{K}}{\mu} \boldsymbol{B}\ d\Omega\ \ with\ \boldsymbol{B} = \nabla \boldsymbol{N} \tag{1.12}$$

and

$$\boldsymbol{q} = \int_\Gamma \boldsymbol{N}^T \frac{\boldsymbol{K}}{\mu}\ (\nabla P \cdot \boldsymbol{n})\ d\Gamma \tag{1.13}$$

Front tracking

The solution of the pressure field with the associated boundary conditions requires the solution over the entire flow domain, which is changing continuously throughout the mold filling process modeling process. To avoid difficulties involving changing flow calculation domains and converging/diverging flow fronts, the Finite Element - Control Volume (FE-CV) method, in which the computational grid is fixed and is based on the mold geometry is employed in this work. This technique uses a single fixed grid throughout the simulation. The FE-CV technique that is employed for the computation of pressures, flow fields, and tracking of flow fronts is briefly described next.

For the present study, three-noded triangle and four-noded quadrilateral elements are employed. The normal finite element shape functions are used in the computation of the finite element matrices of each element. In addition, each node is also associated with a specific control volume region. The finite volume region is obtained by joining the mid points of each of the line segments of the element edges and the body centroid of the elements. Each finite volume region associated with a node has one-fourth of the volume in the case of a quadrilateral element and one-third of the volume in the case of a triangular element. The continually changing flow domain is determined based on the conservation of mass principle applied to each finite volume region. The mass conservation principle is based on the fact that the net mass flow into a finite volume region is equal to the amount of mass absorbed and accumulated in the fiber bundles within the finite volume region. In this formulation, each node in an element is associated with a sub-finite volume region that is local to the element. The total finite volume region associated with a node involves contributions from the elements sharing a given node and can be determined based on the elemental connectivities. The net mass flux entering a finite volume region associated with a node is computed based on the flux at each face of the finite volume region; the computations are based on the pressure gradients and the associated Darcy's velocity field. The net mass flow rate entering the finite volume associated with a node within an ith element is determined from the surface integrals defining the finite volume and can be represented as

$$q_i = \int_S \boldsymbol{u} \cdot \vec{n} dS \tag{1.14}$$

where the velocity field $\boldsymbol{u}$ is determined from the pressure gradients and nodal pressures within the element. The 3-D velocity field is given by

$$u_i = -\nu K_{ij} P_{,j} \qquad i = 1, 2, 3 \tag{1.15}$$

Note that the Darcy's velocity field and the pressure gradients are computed within the element and are continuous within a given element. This implies that the flow rate computed based on the velocity fields

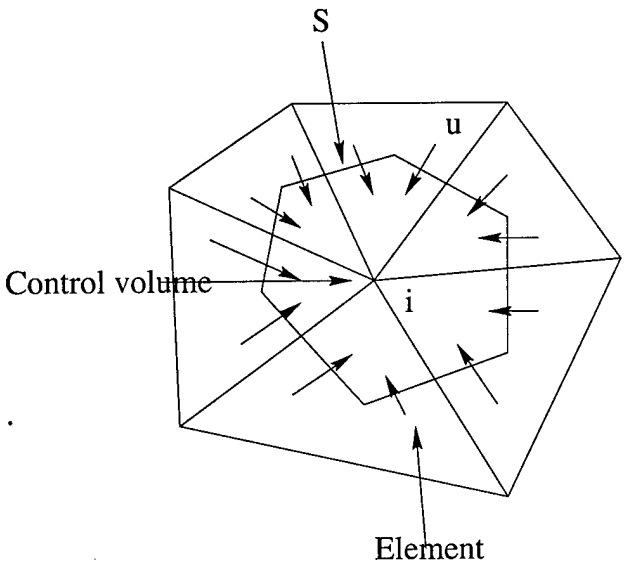


Figure 1.5: Velocity and flow computation for the control volume

satisfies the continuity condition or the local mass conservation condition. This is in contrast to some of the earlier Finite Element/Control Volume techniques in 2-D RTM simulations where the pressure gradients and hence the flow rates were determined across the elements [1–4], even when the elements employed were C_0 continuous. This also permits use of traditional finite elements as opposed to the non-conformal elements as in Trouchu et al. [8] for this purpose.

The continually changing flow front and the filled flow domain is determined from a parameter called the fill factor F_f that is associated with each node. The parameter F_f represents the status of the finite volume and indicates if a given finite volume region is full or empty. If a finite volume region is empty (has not yet been filled with resin), the associated fill factor F_f is zero, and a finite volume that is filled completely with the resin is given a fill factor F_f of 1. The resin front exists in finite volume regions where the fill factor F_f is between 0 and 1, and the factor F_f is set to the volume fraction of the finite volume that is filled with resin. For these nodes with the associated finite volumes, the pressure is zero at the flow fronts and the corresponding nodes. Hence, the solution process in the RTM simulation involves solving for both the pressure field derived from the continuity equation and the factor F_f. Numerous techniques can be employed for the update of the fill factor F_f. The so-called explicit filling techniques determine the fill regions and time based on the actual flow rates and the finite volumes associated with each node. In this work, the advancement of the fill factor, hence, the flow front, is based on an explicit algorithm. In this approach, the mold filling process is regarded as a quasi-steady process, which assumes a steady-state condition at each time step. At each time step, the pressure distribution is computed both based on the boundary conditions and the fill factor conditions. The velocity field, hence, the flow rate computed from the pressure field, is used to update the new front location, fill stage, and time of simulation. The selection of the time step for each of the quasi-steady states is based on the following consideration. At any time, the time increment used allows only one finite volume region to be completely filled (under certain conditions, more than one control volume can be filled). This restriction of the time increment ensures the stability of the quasi-steady state approximation. Thus the computational methodology can be briefly summarized as follows.

Solution strategy

The various computational steps involved in the traditional explicit CV-FE methodology for the update of the filled regions and flow front advancement are summarized below:

1. Compute the local coordinates of elements.

2. Compute the local material properties of elements.

3. Compute control volumes (Ω_{cv}) around all the nodes.

4. Form $\boldsymbol{K_s}$ and $\boldsymbol{q}$ using Eqs.(1.12) and (1.13), respectively.

5. Apply boundary conditions on $\boldsymbol{K_s}$

   - Boundary conditions based on injection, vent pressures, and injection flow rate.
   - Boundary conditions based on the fill factor F. These are applied at nodes where $F < 1.0$.

6. Solve $\hat{K}_{ij} * P_j = q_i$ for pressure P, where $\hat{K}_{ij}$ is the modified $\boldsymbol{K_s}$ after the application of boundary conditions.

7. Compute the local velocity field using Darcy's law.

$$\hat{\boldsymbol{u}} = -\frac{\boldsymbol{K}}{\mu}\nabla P \tag{1.16}$$

   (Pictorially shown in Fig. 1.5 for 3-noded triangular element).

8. Compute the flow rate contributions from each of the surrounding nodes and determine the total flow rate into a control volume region associated with the node (Fig. 1.5).

$$q_i = \int_S \boldsymbol{u} \cdot \vec{n} dS \tag{1.17}$$

9. Determine the time required to fill each of the control volume regions Ω^i_{cv}.

$$\Delta t_i = (1 - F_i) \times \frac{\Omega^i_{cv}}{q_i} \tag{1.18}$$

10. To ensure the stability of the quasi-steady state approximation, select as the time increment the minimum of all the computed times in Step 9. The minimum time step is the quasi-steady state time step increment at this stage and ensures stability with the smallest empty region completely filled during this time increment.

$$\Delta t = min\left[\Delta t_1 \Delta t_2 \ldots \Delta t_n\right] \quad n = number\ of\ nodes \tag{1.19}$$

11. Update the fill factors of the partially filled and unfilled nodal control volume regions using the associated total flow rate and the selected time step increment.

$$F_i = \begin{cases} F_i + \Delta t \times \frac{q_i}{\Omega^i_{cv}} & if\ F_i < 1.0 \\ F_i & otherwise \end{cases} \tag{1.20}$$

12. Continue with the next step of the quasi-steady state process until filling is complete. That is, until all the control volumes have the fill factor equal to 1.

1.2.3 Implicit filling technique: A Pure Finite Element Methodology

The impregnation of the porous fiber medium during an RTM process is realistically a physical transient problem involving time dependent progression and distribution of the resin mass. The control volume-finite element methodology treats this problem as a quasi-steady state problem involving steady state solutions at discrete time intervals. The physical accuracy and computational effectiveness of the CV-FE modeling and simulation methodology are thus restricted by the quasi-steady state assumption and the subsequent time step restrictions required to ensure stability of the methodology. Furthermore, large finite element meshes and finer computational grids can heavily restrict the time step increments that can be employed and

significantly impact the computational speed and costs for real time process simulations. It is imperative that the computational methodologies employed for the physical modeling and simulations be physically accurate and computationally efficient to provide for further enhancements towards faster real time process simulations. Driven by the needs for improved physical accuracy and faster computational speed of the process simulations, a pure finite element methodology is employed here based on an alternate form of the mass conservation equation [5, 6].

Conservation of resin mass

In a mold-filling analysis, a fill factor can be implicitly defined to represent the amount of resin present inside the mold and its distribution at any time. Considering for a moment a general Eulerian mold domain Ω with mass flux due to the incoming resin at the injection ports, at any instant, part of the mold cavity is filled with resin and the fill factor Ψ is greater than zero. In the unfilled regions, there is no resin and the fill factor Ψ is equal to zero. For convenience, the maximum value of the fill factor Ψ is set to be one.
At any instant in time, t, the mass of the resin inside the mold is given by

$$\text{mass} = \int_{\Omega} \rho \Psi(t)\, d\Omega \tag{1.21}$$

where ρ is the density of the polymer resin and the definition of Ψ which is the fill factor considers only the mold cavity filled with resin. From the continuity equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho\, \boldsymbol{v}) = 0 \tag{1.22}$$

the mass conservation equation at any instant of time for the resin is written as:

$$\frac{\partial m}{\partial t} = \frac{\partial}{\partial t} \int_{\Omega} \rho \Psi(t)\, d\Omega = - \int_{\Gamma} \rho \Psi(t) \boldsymbol{u} \cdot \boldsymbol{n}\, d\Gamma \tag{1.23}$$

Assuming a constant density resin, the modified mass conservation equation for the resin at any instant of time reduces to

$$\frac{\partial}{\partial t} \int_{\Omega} \Psi(t)\, d\Omega = - \int_{\Gamma} \Psi(t) \boldsymbol{u} \cdot \boldsymbol{n}\, d\Gamma \tag{1.24}$$

Eq.(1.24) now forms the starting point for the development of the pure finite element methodology.

Governing equations

For the polymer resin flowing through a porous media as in RTM flows, the velocity field is governed by Darcy's flow field approximation and is dependent on the permeability of the resin and the resin viscosity. This velocity field is given by

$$\boldsymbol{u} = -\frac{\boldsymbol{K}}{\mu} \nabla P \tag{1.25}$$

where $\boldsymbol{K}$ is the permeability tensor of the fiber preform and is defined appropriately for two-dimensional and three-dimensional preform considerations. Using Darcy's approximation in the mass balance equation, Eq.(1.24) can be written as

$$\frac{\partial}{\partial t} \int_{\Omega} \Psi(t)\, d\Omega = \int_{\Gamma} \Psi(t) \left(\frac{\boldsymbol{K}}{\mu} \nabla P \right) \cdot \boldsymbol{n}\, d\Gamma \tag{1.26}$$

Employing Green's theorem, which states that

$$\int_{\Omega} \text{div}\, F = \oint_{\Gamma} F \cdot \boldsymbol{n}\, d\Gamma \tag{1.27}$$

the mass balance equation involving the fill factor and the pressure field is obtained and given as

$$\frac{\partial}{\partial t} \int_{\Omega} \Psi(t)\, d\Omega = \int_{\Gamma} \Psi(t) \left(\frac{\boldsymbol{K}}{\mu} \nabla P \right) \cdot \boldsymbol{n}\, d\Gamma = \int_{\Omega} \Psi(t) \nabla \cdot \left(\frac{\boldsymbol{K}}{\mu} \nabla P \right) d\Omega \tag{1.28}$$

This form of the mass balance equation involving the fill factor and the pressure field is the governing mathematical equation for the present finite element formulations. Since the pressure gradients are negligible in the unfilled and the partially filled regions (i.e., $\Psi < 1$), only the completely filled regions where $\Psi = 1$ are needed to be considered in the governing mass balance equation. With this consideration, the differential form of Eq.(1.28) is

$$\frac{\partial \Psi}{\partial t} = \nabla \cdot \left(\frac{\boldsymbol{K}}{\mu} \nabla P \right) \tag{1.29}$$

The boundary and initial conditions for the mass balance equation are:

At mold surface:	$\frac{\partial P}{\partial n} = 0$ on Γ_1
At resin front:	$P = 0$ on Γ_1
At mold inlet:	$P = P_0$ (prescribed pressure), or
	$q = q_0$ (prescribed flow rate)
At injection ports:	$\Psi(t \geq 0) = 1.0$

Space discretization

To determine the pressure field and the resin saturation fill factors, the mold cavity is discretized and modeled using finite elements. For thin shell molds, two-dimensional elements with two-dimensional velocity fields and two-dimensional permeabilities are employed. For thick composite sections, a three-dimensional velocity field as given by Darcy's law and involving a three-dimensional permeability tensor is used. Thus, the present formulations are applicable for both thin and thick composites. Invoking the traditional Galerkin weighted residual formulation, Eq.(1.29) is written as

$$\frac{\partial}{\partial t} \int_\Omega W \Psi \, d\Omega = \int_\Omega W \left(\nabla \cdot \frac{\boldsymbol{K}}{\mu} \nabla P \right) d\Omega \tag{1.30}$$

If the weight W is taken to be the same as the spatial shape function N, and both the pressure and fill factors be approximated by the standard finite-element approximations as given by:

$$\begin{aligned} P &= N_i \, P_i \\ \Psi &= N_i \, \Psi_i \end{aligned} \tag{1.31}$$

Eq.(1.30) reduces to

$$\frac{\partial}{\partial t} \left[\int_\Omega N_i N_j \, d\Omega \right] \Psi_i = \left[- \int_\Omega \nabla N_i \frac{\boldsymbol{K}}{\mu} \nabla N_j \, d\Omega \right] P_i + \int_\Gamma N_i \left(\frac{\boldsymbol{K}}{\mu} \nabla P \cdot \boldsymbol{n} \right) d\Gamma \tag{1.32}$$

where N_i are the spatial shape functions at each node of the finite element and P_i and Ψ_i are the nodal values of the pressure and the fill factor, respectively. The terms on the right hand side of Eq.(1.32) were obtained invoking the Green-Gauss theorem and the approach leads to a pure finite element based technique without the notion of having control volume regions to be associated in a given finite element mesh.
The discretized system of equations can also be written as

$$\boldsymbol{C} \dot{\boldsymbol{\Psi}} + \boldsymbol{K_s} \boldsymbol{P} = \boldsymbol{q} \tag{1.33}$$

where

$$\boldsymbol{C} = \int_\Omega \boldsymbol{N}^T \boldsymbol{N} \, d\Omega \tag{1.34}$$

$$\boldsymbol{K_s} = \int_\Omega \boldsymbol{B}^T \frac{\boldsymbol{K}}{\mu} \boldsymbol{B} \, d\Omega \tag{1.35}$$

and

$$\boldsymbol{q} = \int_\Gamma \boldsymbol{N}^T \left(\frac{\boldsymbol{K}}{\mu} \nabla P \cdot \boldsymbol{n} \right) d\Gamma \tag{1.36}$$

where $\boldsymbol{B}$ are the spatial derivatives of the shape functions employed in the finite element discretization.

Introducing the finite difference approximation for the time derivative term in Eq.(1.33), where

$$\dot{\boldsymbol{\Psi}} = \frac{\boldsymbol{\Psi}^{n+1} - \boldsymbol{\Psi}^{n}}{\Delta t} \tag{1.37}$$

the discretized finite-element system is written as

$$\boldsymbol{C}\left[\boldsymbol{\Psi}^{n+1} - \boldsymbol{\Psi}^{n}\right] + \Delta t \boldsymbol{K_s P} = \Delta t \boldsymbol{q} \tag{1.38}$$

For a lumped $\boldsymbol{C}$, Eq.(1.38) reduces to

$$C_{ii}\Psi_i^{n+1} - C_{ii}\Psi_i^{n} + \Delta t K_{ij} P_j = \Delta t\ q_i \tag{1.39}$$

Solution strategy

Since Eq.(1.39) is the discretized model equation in the present methodology, the fill factors associated with the nodes and the pressure field are solved for in an iterative manner. At the beginning of the simulation, the fill factors are known and are taken to be unity at the injection nodes. Then at each time step, values of the pressure field and the fill factors are iterated until mass conservation is reached. Below is a summary of the iterative procedure used in the implicit Pure FE methodology:

1. Compute the local coordinates of elements.
2. Compute the local material properties of elements.
3. Compute the matrix $\boldsymbol{C}$ using Eq.(1.34) and sum the off-diagonal entries of each row with the diagonal entry of $\boldsymbol{C}$ to form the lumped matrix of $\boldsymbol{C}$.
4. Form $\boldsymbol{K_s}$ and $\boldsymbol{q}$ using Eqs.(1.35) and (1.36), respectively.
5. At the beginning of the each time step,
$$(\Psi_i)_m^{n+1} = (\Psi_i)^n \tag{1.40}$$
where m is the iteration number and $n+1$ and n are the current and previous time steps, respectively.
6. Apply boundary conditions on $\boldsymbol{K_s}$
   - Boundary conditions based on injection, vent pressures, and injection flow rate.
   - Boundary conditions based on the fill factor F. These are applied at nodes where $\Psi_i < 1.0$.
7. Form
$$(g_i)_m = C_{ii}(\Psi_i)^n - C_{ii}(\Psi_i)_m^{n+1} + \Delta t q_i \tag{1.41}$$
8. Solve $\hat{K}_{ij}(P_j)_m = (g_i)_m$ where $\hat{K}_{ij}$ is the modified $\boldsymbol{K_s}$ after the application of boundary conditions.
9. Update the nodal resin fraction field Ψ_i^{n+1} using the modified form of the discrete mass balance equation
$$C_{ii}\Psi_i^{n+1} = C_{ii}\Psi_i^{n} - \Delta t K_{ij} P_j + \Delta t q_i \tag{1.42}$$
Note that only a matrix vector product and vector additions are involved here.
10. Correct for under filling or over filling. Since Ψ can be either greater than 1 or less than 0, a correction needs to be made.
$$(\Psi_i)_{m+1}^{n+1} = \max\left[0, \min\left(1, (\Psi_i)_m^{n+1}\right)\right] \tag{1.43}$$
Note that the bounds imposed on the fill factor are $(0 \leq \Psi \leq 1)$ and mass conservation check is imposed at every iteration in a time step. The fill factor Ψ and pressure $\boldsymbol{P}$ are related through:
$$\begin{aligned} &if \quad \Psi_i < 1, \quad p_i = 0 \\ &if \quad \Psi_i = 1, \quad p_i = p_i^{calculated} \end{aligned} \tag{1.44}$$
11. Continue until convergence is reached,
$$\left\| C_{ii}\Psi_{i}{}_{m+1}^{n+1} - C_{ii}\Psi_{i}{}_{m}^{n+1} \right\| \leq \epsilon \tag{1.45}$$
12. Proceed to next time step.

1.3 Parallel algorithm and data structures

In the following sections the parallel implementation of the above two approaches on a shared memory model and message-passing model are discussed. Linear iterative solvers based on conjugate gradient approaches are employed in this study.

1.3.1 A frame work for the parallel finite element implementation

Most of the computations are computed at the element level. These include the formation of element stiffness matrix, computation of sub-control volumes and control volume flowrates (in the case of explicit CV-FE). These operations can be, in principle, carried out in parallel with all the processors in a single node without any synchronization. However once this task is accomplished, these data should be accumulated to the nodal level. Depending upon the programming model used and the parallel architecture, different possibilities exist for the above mentioned task. This is also the interface module to be used in the 'old' serial code for the portable general parallel finite element analysis programs. Three different algorithms are discussed in the following sections.

Shared memory model with element based partition

Elements are distributed across the processors as shown in Fig. 1.6. Each processor computes the nodal contribution of all the nodes assigned to it with all the elements shared by a finite element node in that processor. For the interface nodes the processor is required to collect information of the nodal contribution from the neighboring processor. For this an efficient shared memory algorithm is described next with reference to the mesh in Fig. 1.6.

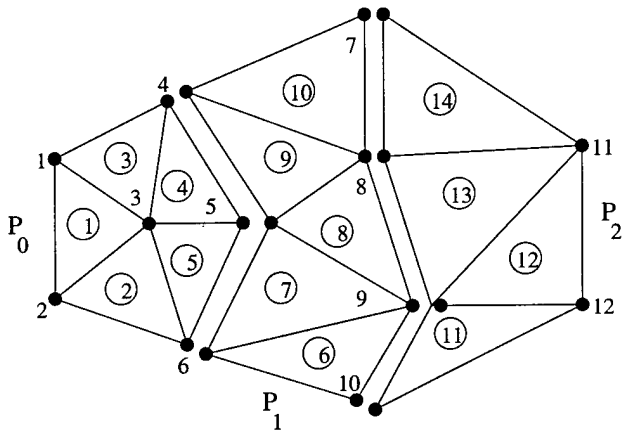


Figure 1.6: Finite element mesh partitioned into three sub-domains

The variable `myid` has the id of the processor. The variable `numadd` stores the number of processors that this processor needs to get the information. The actual ids of the neighboring processors are stored in `addlist`. The subroutine writeReadIBV; IBV, Interface Boundary Variable, does the communication between the processors for the variable shared between the sub-domains. The arrays is of the size `numadd`. The array `writeReadPtr` and `writeReadInd` store the information of `IBValues`. `IBValues` is the array which holds the values of the interface variable such as control volume, residual vector, control volume flowrates, lumped matrix C and the like. The array `IntVariable` is the shared buffer used to write and read the `IBValues` by all the processors. It is a 2D array with number of columns equal to the number of processor `nproc` and number of rows equal to the maximum number of the interface nodes, `max(writeReadPtr(i+1)-writeReadPtr(i), i=1,2,..nproc`. Fig. 1.7 shows the values of the data structure for the finite element mesh shown in Fig. 1.6. The following subroutine `writeReadIBV` does the communication with the neighboring processors.

```
1       Subroutine writeReadIBV(myid,numadd,writeReadPtr,writeReadInd,addlist,
```

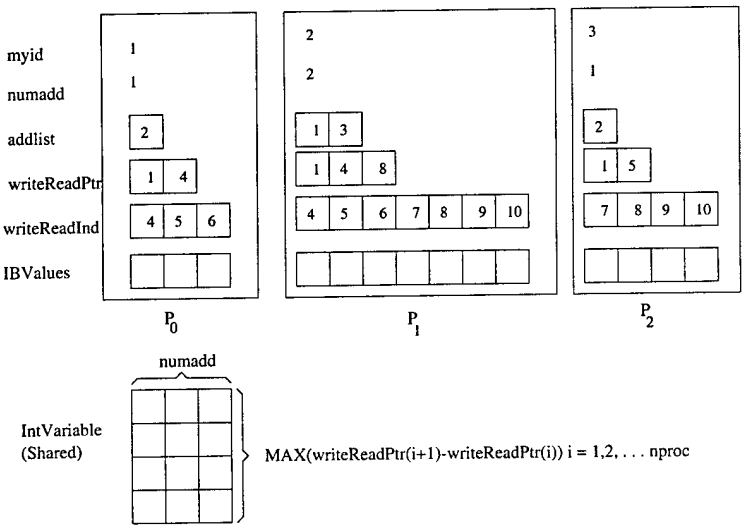


Figure 1.7: Values of the data structure for the partitioned finite element mesh Fig 1.6

```
2                              IBValues,nproc)
3       Real    IBValues(1), IntVariable(1000,12)
4       Integer writeReadPtr(1), writeReadInd(1)
5       Integer addlist(1)
6       Common /IBV/IntVariable
```

At this point all processors write the shared variable values to the shared buffer `IntVariable`.

```
7       Do 10 i = 1, numIBV
8          IntVariable(i,myid) = IBValues(node)
9 10    Continue
```

Wait for all the processor to write the information to the shared buffer.

```
10      Call barrier(nproc)
```

Now update the interface variable with the information in the shared buffer.

```
11      Do 20 procId = 1, numadd
12            start = writeReadPtr(procId)-1
13            Do 30 i = 1, nIBV
14               node = writeReadInd(start+i)
15               IBValues(node) = IBValues(node) + IntVariable(i,procId)
16 30         Continue
17 20   Continue
```

Message-passing model with element based partition

A similar algorithm discussed previously for the message-passing model is discussed next with reference to the mesh in Fig. 1.6

The variable `myid` has the id of this node. The variables `numadd` store the number of nodes that this node needs to send and receive data. These are the neighboring nodes. The actual ids of the neighboring

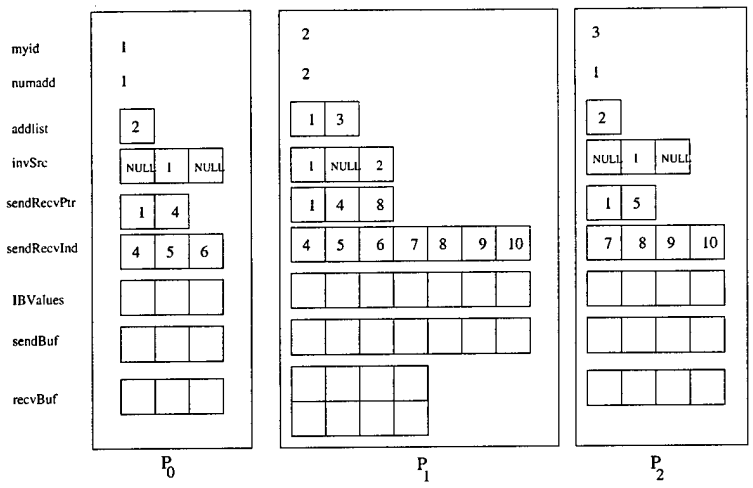


Figure 1.8: Values of the data structure for the partitioned finite element mesh Fig 1.6

nodes are stored in the arrays `addList` . The array is of the size `numadd` . Note that if the data is sent to a neighboring node, then this node needs to receive data from that neighboring node. The arrays `sendRecvPtr` and `sendRecvInd` store the information of the `IBValues`. The indices of the elements that are sent to the i^{th} neighboring node are stored in `sendRecvInd` starting at location `sendRecvInd(i)` and ending at location `sendRecvInd(i)-1`. The array `sendRecvPtr` is of the size `(numadd+1)` and size of the array `sendRecvInd` is `sendRecvPtr(numaddl)-1+`. The arrays `sendBuf` and `recvBuf` are used as buffers to store the values of `IBValues` that are sent and received. The size of the array `sendBuf` is same as `sendRecvInd`. The array `recvBuf` is a 2D array where the number of columns equal to `numadd` and the size of the column is equal to maximum of `sendRecvPtr(i) - sendRecvPtr(i) for i = 1,2.. nproc`. Figure 1.8 shows the values of the data structures for the finite element mesh shown in Fig 1.6. For illustration the values of the variable `sendRecvInd()` shown in Fig. 1.8 are the actual node numbers of the mesh in Fig. 1.6. In the actual program, the renumbered node numbers are used.

The following message passing subroutine, `SendRecvIBV()`; IBV, Interface Boundary Variable, does the communication between the neighboring nodes. The arrays `request`, `status` and `indx` are temporary arrays required for the `MPI` libraries.

```
1       Subroutine SendRecvIBV(myid, numadd, sendRecvPtr, sendRecvInd,
2      .                       addlist, invadd, IBValues, mlen )
3       Include 'mpif.h'
5       Real    IBValues(1), sendBuf(1000), recvBuf(1000,64)
6       Integer sendRecvPtr(1), sendRecvInd(1)
7       Integer addlist(1), invadd(1)
8       Integer request(64), status(MPI_STATUS_SIZE), recvSrc(64)
```

At this point pack the send buffer `sendBuf` with the interface boundary variable values `IBValues` for the all the neighboring nodes.

```
9       Do j = sendRecvPtr(1), sendRecvPtr(numadd+1)-1
```

```
10          node = sendRecvInd(j)
11          sendBuf(j) = IBValues(node)
12      Enddo
```

Send all the sendBuf information to the neighboring nodes using MPI_ISEND(). MPI_ISEND() is a non-block send routine. The sending node is not blocked until the corresponding receive is started. With the use of this routine all the data transfer to the neighboring nodes can be initiated.

```
13      tag = myid + 1
14      Do i = 1,numadd
15         call MPI_ISEND(sendBuf(sendRecvPtr(i)),len(i),MPI_REAL,
16     .                  addlist(i)-1,tag,MPI_COMM_WORLD,request(i),IERR)
17      Enddo
```

Receive the information from the neighboring nodes. Since the network load and neighboring computational load is not known, this node does not know from whom it will receive the information first. To achieve this MLEN, MPI_ANY_SOURCE and MPI_ANY_TAG variables are used in the MPI_RECV() command. Also there is no need to use non-block receive which will increase the uncertainty. The id of the node from whom the data is received is stored in recvSrc array.

```
18      Do i = 1, numadd
19         Call MPI_RECV(recvBuf(1,i),MLEN,MPI_REAL,MPI_ANY_SOURCE,
20     .                 MPI_ANY_TAG,MPI_COMM_WORLD,request(i),IERR)
21         recvSrc(i) = status(MPI_SOURCE) + 1
22      Enddo
```

At this point wait for all the non-block communication to complete. This will ensure that the communication is safely complete.

```
23      Call MPI_WAITALL(request,indx,status,IERR)
```

Once the communication is over, then update the interface boundary variable which has been received from the neighboring nodes.

```
23      Do 101 i = 1,numadd
24             isource  = recvSrc(i)
25             invsrc   = invadd(isource)
26             istart   = sendRecvPtr(invsrc) - 1
27             Do 110 j = 1, len(invsrc)
28                    node = sendRecvInd(istart+j)
29                    IBValues(node) = IBValues(node) + recvBuf(j,i)
30  110        Continue
31  101 Continue
32      Return
33      End
```

Shared memory model without any type of partition

If the elements are not partitioned then all the processors can process all the elements. Since nodal level data is also shared, there exists a race condition and hence the expensive critical regions. Critical region

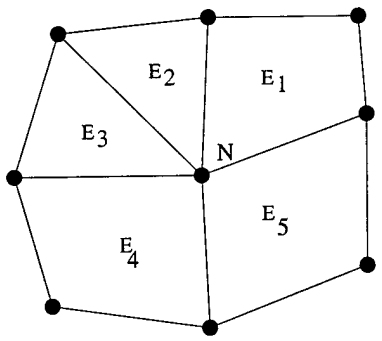


Figure 1.9: A single node shared by five elements

computations are serialized at this level and hence significantly degrade performance. In Fig. 1.3.1 a mesh node N is attached to five distinct elements $E_1 \ldots E_5$. If NV is the nodal quantity and is the accumulation of independent elemental contributions, UE^i a segment of FORTRAN code that performs this operation would be:

```
1       NV = 0.0
2       Do 10 EleId = 1,5
3          EV = function(EleId,...)
4          NV = NV + EV
5       10 Continue
```

In this case, if all the elements are assigned to different processors by index $EleId$, then each processor will have the private copy of UE , and will be evaluated in parallel. But all the processors cannot update the value NV simultaneously. This section of the code is called a critical region and an explicit synchronization is required. This is done as follows.

```
1       NV = 0.0
2       Do 10 EleId = 1,5
3          EV = function(EleId,...)
4         Call lock()
5               NV = NV + EV
6          Call unlock()
7 10    Continue
```

`lock` and `unlock` constructs guarantee that only one process at a time will execute the block of code between these two constructs. Thus the computation is serialized at this level and each processor sequentially updates the value of NV.

However this can be avoided by processing the elements in a specific order. These specific orders are created by coloring the elements and processing each list concurrently with synchronization points at the end of each color list. The segment of the code for above computation can be written as follows.

```
1       NV = 0.0
2       Do 10 colorId = 1, numColor
3          Do 20 indx = colorPtr(colorId), colorPtr(colorId+1)-1
```

```
4               EleId = eleList(indx)
5               EV = function(EleId,...)
6               NV = NV + EV
7 20        Continue
8       Call Barrier(nproc)
9 10    Continue
```

where the inner loop is parallelized. `numColor` is the number of colors resulted due to coloring of the finite element mesh. It is also the number of synchronization points in the outer loop. Therefore the coloring algorithm should result in few colors as possible. `numele` is the number of elements and `eleList` are arrays having number of elements and element id list for `colorId`. `Barrier` construct provides the synchronization, where all the processors wait until every processor arrives at that point.

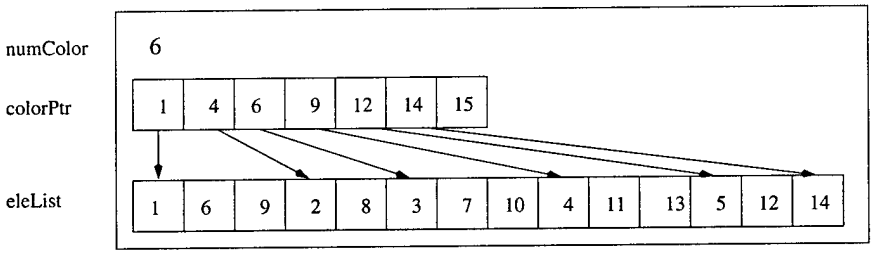


Figure 1.10: Values of data structure for the coloring of finite element mesh Fig. 1.6

Figure 1.3.1 shows the values stored in the data structure for the finite element mesh Fig. 1.6.

1.3.2 Shared memory model implementation of mold filling

The section of the serial code of the main program where application subroutine is called would resemble as follows.

```
1       Program MAIN
2       Call getModelData(....)
3       Call applicationSubroutine(....)
4       Return
5       End
```

where `applicationSubroutine` could be the explicit control volume-finite element algorithm subroutine or the implicit pure finite element algorithm subroutine. In the parallel implementation, the entire domain is decomposed into sub-domains to equal the number of processors. Each of the sub-domains will have approximately an equal number of elements. Once the sub-domains are formed, data structures for inter processor communication as explained in previous section are created. The sub-domains are then mapped on to the processors. This is done by creating the number of process (sub-domains) equal to the number of processors with the help of `createThread` routine. The section of the code of main program is given below. If the SGI constructs are used, then `creatThread` routine is called `sproc()` and if POSIX (pthreads) standard is used then the routine is called `pthread_create()`. One of the arguments passed to this routine is the name of the application subroutine (explicit CV-FE or implicit Pure FE) `applicationSubroutine`.

```
1       Program MAIN
2       Call getModelData(nproc,....)
3       Call createSubDomains(nproc,....)
4       Call initializeThreads()
5       Do 10 procId = 2, nproc
6          createThread(applicationSubroutine,procId)
7  10   Continue
8       Call applicationSubroutine(1)
9       Return
10      End
```

Explicit control-volume finite element method

Steps 1–5, of the explicit control-volume finite element mold filling algorithm listed in section 4.2.2 is performed asynchronously by each processor. In step 3, once the control volumes of all the nodes in the sub-domain is computed by each processor, the subroutine `writeReadIBV()` is called to gather control volume information of the interface nodes from the neighboring sub-domains. Step 6 is accomplished by the conjugate gradient solver explained in chapter 3. Interior control volume flow rates are computed by each processor, as in step 7, 8. Once again the routine `writeReadIBV()` is called to gather the interface nodes flow rates. In steps 9 and 10, each processor computes the Δt_i and the minimum of all of its nodes. Then the global minimum of Δt is computed using the data structure shown in Fig 1.11. Once Δt is computed, the fill factors F are updated without any communication between the processors and steps 5–11 are repeated until all the control volumes have the fill factor equal to 1.

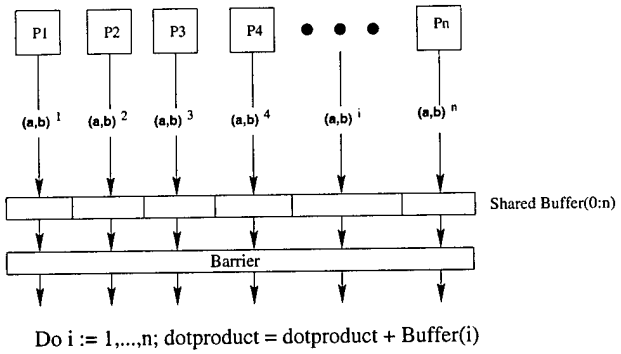


Figure 1.11: Data structure and process synchronization for dot product

Implicit pure finite element method

Steps 1–7, of the implicit pure finite element mold filling algorithm listed earlier are performed asynchronously by each processor. In step 3, once the lumped C of all the nodes in the sub-domain is computed by each processor, the subroutine `writeReadIBV()` is called to gather the lumped C information of the interface nodes from the neighboring sub-domains. Step 8 is accomplished by the conjugate gradient solver explained in chapter 3. Step 9 involves calculating the residual $q_i - K_{ij}P_j$. This is accomplished by computing the

residual for all the nodes at the sub-domain level and using the routine `writeReadIBV` to gather neighboring sub-domain residuals. Now steps 9, 10 and 11 can be completed without any communication. Steps 6–12 are repeated till Eq. 1.45 is satisfied.

1.3.3 Message-passing model implementation for process modeling

The entire domain is decomposed into sub-domains equal to the number of processors. Each of the sub-domains will have approximately an equal number of elements. Once the sub-domains are created with the appropriate data structures for inter processor communication, the sub-domains are then mapped on to the processors. This is accomplished by `MPI_INIT()` construct provided by MPI. A typical message-passing program consist of a single executable that runs on each processor of the parallel computer. The section of the code of the main program is given below.

```
1       Program MAIN
2       CALL MPI_INIT(ierr)
3       CALL MPI_COMM_RANK(MPI_COMM_WORLD, myid,  ierr)
4       CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nproc, ierr)
5       Call getModelData(nproc,....)
6       Call createSubDomains(nproc,....)
7       Call applicationSubroutine(myid,....)
8       CALL MPI_FINALIZE(ierr)
9       Return
10      End
```

Explicit control volume-finite element and implicit pure finite element methods

The implementation is similar to that of the shared memory model except that in place of subroutine `writeReadIBv()`, subroutine `sendRecvIBV()` is used, and instead of the data structure explained in Fig. 1.11, the MPI routine `MPI_ALLREDUCE()` is used with appropriate operation of `MPI_SUM` or `MPI_MIN`.
The important point to note here is that the 'old' serial subroutine code of the simulation is executed for each sub-domain. The usual do loop in the finite element simulation code consists of the following structure.

```
Do eleId = 1, nelments
    computation .....
Enddo
```

For the same 'old' serial subroutine to be used in parallel implementation, all the elements and nodes in the sub-domains are renumbered such that the numbers start from `1` to `nelemnts` where `nelemnts` is equal to number of elements in each sub-domain. This is done before calling the `applicationSubroutine()`. The necessary communication required between the sub-domains are written in the separate module, `writeReadIBV()` for the shared memory model and `sendRecvIBV()` for the message-passing model, which is used throughout the the finite element simulation code. This interface module is a very small portion of the code which can optimized for a given parallel computer. Thus this approach enforces data locality and therefore suitable for

a wide range of parallel architectures resulting in portability and also good performance on a wide range of parallel architectures.

1.4 Closure

Flow modeling computational approaches for the process modeling by RTM were discussed. These include the traditional explicit control-volume finite element methodology and the implicit pure finite element scheme. Data structures and interprocess communication models were also discussed. An efficient scheme is provided in a separate module (subroutine) for sub-domain interface solution for general parallel finite element analysis for a wide range of parallel architectures. This approach preserves data locality and hence high performance is achievable. Both the thread based model and the message-passing model with MPI were given. Also, a parallel programming model for effectively reusing the existing program of one processor which requires only minimum modification is given.

Performance Results

2.1 Introduction

On a sequential computer, the fastest algorithm for solving a given problem is the best algorithm (besides being the most accurate). However, the performance of a parallel algorithm for a specific problem on a given number of processors provides only limited information. The time taken by a parallel algorithm to solve the problem for instance, depends on the problem size, number of processors used to solve the problem and the machine characteristics such as: processor speed, speed of communication channels and the type of interconnection network. An algorithm that yields good performance for a selected problem on a fixed number of processors on a given machine may perform poorly if any of these parameters are changed. Hence, more than one performance metric, elapsed time is required to characterize the performance of parallel systems. There are many ways to measure the performance of parallel systems. The most commonly used measurements are the elapsed time, price/performance, speedup and efficiency. Additionally, researchers have used scalability and experimentally measured serial fraction as the performance metrics for parallel systems. These performance metrics are defined and the significances are discussed in the next section.

2.2 Performance metrics

2.2.1 Elapsed time T_p

The elapsed time to run a particular program on a given machine is the most important metric. It is the time taken by the parallel program from the point of start of its execution till all the processors finish execution. The parallel computation has no value in the industrial setting if it is not applicable to industrial design problems in highly complex and completive product development cycles. With short product design cycles, the effect of parallel computation can only be realized if and only if the solutions can be found within the appropriate time window of the design cycle. Hence, to achieve this the elapsed time should be smaller than

the *Reasonable Time.*

2.2.2 Speedup S

When measuring a parallel system, we are often interested in knowing how much performance gain is achieved by parallelizing a given application over a sequential implementation. Speedup is a measure that computes the relative benefit of solving the problem in a parallel computer. Speedup is defined as the ratio of serial execution time of the fastest known serial algorithm, T_s, to the parallel execution time of the chosen algorithm T_p. Mathematically, it is given by

$$S = \frac{T_s}{T_p} \tag{2.1}$$

However, if the interest lies in studying the parallel algorithms for parallel systems, T_s can be the serial execution time of the algorithm on the parallel computer on which the study is made. Ideally, speedup attained should be equal to the number of processors p used in the application.

2.2.3 Efficiency E

The efficiency, E is related to that of price/performance. It is defined as the ratio of speedup, S, to the number of processors P. The mathematical expression is given by

$$E = \frac{S}{p} = \frac{T_s}{p\,T_p} \tag{2.2}$$

An efficiency close to unity means that the hardware has been used effectively; a low efficiency means that the resources are being wasted.

2.2.4 Experimentally measured serial fraction f

Experimentally measured serial fraction, f, is defined as [9],

$$f = \frac{\left(\frac{1}{s} - \frac{1}{p}\right)}{\left(1 - \frac{1}{p}\right)} \tag{2.3}$$

This metric is mainly used as the diagnostic tool to find the reasons for loss of performance in speedup and efficiency. The value of f is exactly equal to the serial fraction s of the algorithm if the loss in speedup is only due to serial component (i.e if the parallel program has no overheads). Smaller values of f are considered better. Following are the conclusions that can be drawn from the the values of f.

- If f increases with the number of processors, then it is considered as an indicator of rising communication overhead.
- If the value of f decreases with increasing p, then following Karp and Flatt [9] it is an anomaly which explains the phenomena such as super-linear speedup effects or cache effects.

- If there is irregular change in the values of f as p increases then it indicates that load imbalance exists and hence the reason for loss of performance.
- If the serial fraction is nearly constant for all values of p, then loss of efficiency is due to the limited parallelism of the program.

2.2.5 Scalability

It is a well known fact that given a parallel architecture and a problem of fixed size, the speedup of a parallel algorithm does not continue to increase with increasing number of processors. It usually tends to saturate or peak at a certain limit. Thus it may not be useful to employ more than an optimal number of processors for solving a problem on a parallel computer. This optimal number of processors depends on the problem size, the parallel algorithm and the parallel architecture.
One way of expressing scalability of a parallel system is the increase in the problem size required in order to maintain the efficiency at a fixed value for increasing P. It can be expressed as a function called an isoefficiency function mathematically expressed as [10]

$$W = f_E(P) \tag{2.4}$$

where W is the problem size. The important feature of isoefficiency analysis is that in a single expression, it succinctly captures the effects of the characteristics of a parallel algorithm as well as the parallel architecture on which it is implemented. By performing the isoefficiency analysis one can test the performance of a parallel program on a few processors and then predict its performance on a larger number of processors. Ideally isoefficiecy functions should be linear for a highly scalable parallel system. For details of isoefficiency analysis one is referred to [10].

2.3 Results: Performance Study

All the computations are performed on the following: (i) SGI Power Challenge which is a symmetric multiprocessor and cc-NUMA machine of the type shown in Fig. 2.1.b. The similar machines in this category are (i) Encore Multimax, Flex/32, Sequent Balance and Alliant FX/8, and (ii) SGI Origin2000 which is also a symmetric multiprocessor and cc-NUMA machine of the type shown in Fig. 2.1.c. The similar machines in this category are KSR-1, TC2000, and Stanford DASH. The message passing is carried out using Message Passing Interface (MPI) and thread programming was accomplished by SGI provided constructs. The Multilevel k-way partitioning [11] algorithm is used for partitioning the finite element meshes into submeshes. The performance study of the equation solver and the adapted approaches, namely, the explicit control volume-finite element and the implicit pure finite element mold filling studies on different meshes of different sizes are carried out. The details of four different highly unstructured meshes involving increasing

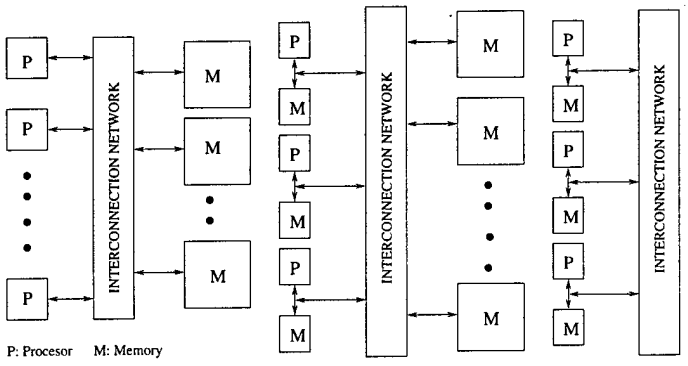


Figure 2.1: A typical shared-address-space architectures. (a) UMA shared-address-sp ace computer (b) NUMA shared-address-space computer with local and global memories (c) NUMA shared-address-space computer with local memory only. [1]

number of elements, nodes and number of non-zero entries in the stiffness matrix are shown in Table. 2.1. For the implicit pure FE RTM mold filling approach the time step size chosen (so as to provide a comparable resolution of the flow fronts as the explicit CV-FE) are listed in Table. 2.2.

Geometry	Nodes	Ratio	Elements	Ratio	Non-zeros in $\boldsymbol{K}$	Ratio
MESH1	4,380	1.0	8,670	1.0	30,482	1.0
MESH2	45,547	10.4	89,945	10.4	316,593	10.4
MESH3	135,492	30.9	269,835	31.1	946,208	31.0
MESH4	297,576	67.9	594,756	68.6	2,082,302	68.3
MESH5	405,327	92.5	809,505	93.4	2,835,053	93.0

Table 2.1: Details of composite structure mesh geometries under study

2.3.1 Evaluation of communication characteristics

It is necessary to evaluate the interprocess communication characteristics which include finding parameters t_s and t_w, where t_s is the start up time for transferring a message and t_w is the per byte transfer time between the two processors and to confirm the the assumed interconnection network in the algorithm complexity Eq. 2.8. These parameters are used in predicting the performance of the proposed algorithm (PCG) given in Chapter 3. The t_s and t_w are evaluated as follows. Communication time for point-to-point communication between the two processors can be modeled by

$$T(n) = t_s + t_w n \tag{2.5}$$

Geometry	fill time (sec)	time step Δt
MESH1	680.0	1.0
MESH2	52.0	2.0
MESH3	5.2	0.2
MESH4	18.2	1.3
MESH5	6.5	1.3

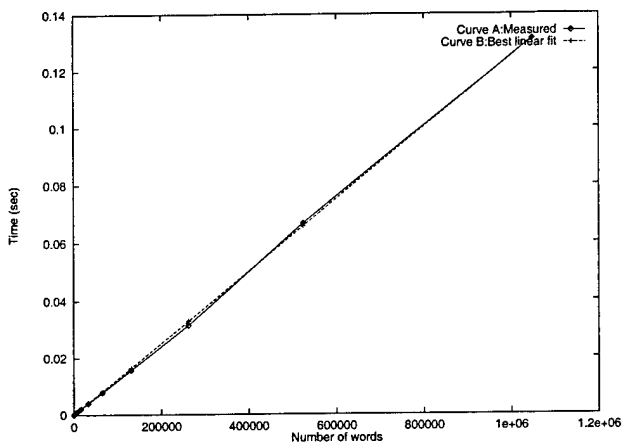
Table 2.2: Fill time and time step for implicit pure FE approach

where n is the number of bytes transfered. In the first test series, an integer (4 bytes) array with increasing size (size $= 2^i$ $i = 0, 1, \ldots, 20$) was transfered. The test was carried out on the SGI Origin2000 machine. Communication time $T(n)$ and message size pair is plotted as shown in Curve A of Fig 2.2.a. The Curve A confirms the hypothesis made in Eq. 2.5. The parameters t_s and t_w found using the best linear fit to the Curve A and the best linear fit curve is plotted as Curve B in Fig 2.2.a. The parameter t_w is the slope of Curve B and the parameter t_s is computed by computing $T(0)$. The computed values are found to be t_w =1.125×10^{-8} sec and t_s =4.6×10^{-5} sec. Thus Eq. 2.5 can be written as

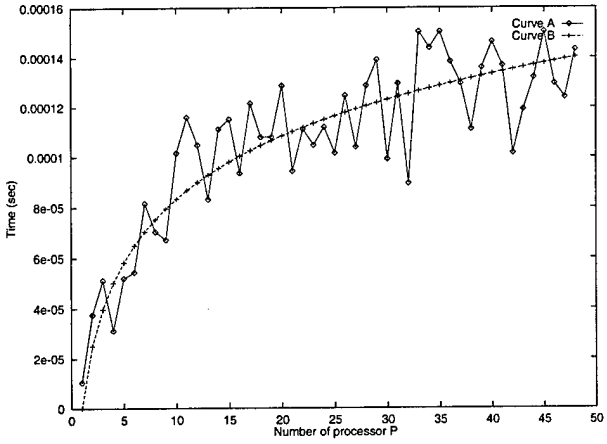
$$T(n) = 10^{-8}(4600 + 1.125\,n) \tag{2.6}$$

Since message-passing is carried out on shared-address space architecture, the figures of t_s and t_w can be compared with the reported peak memory latencies of SGI Origin2000 [12]. The message passing latency (t_s) equal to 46 milli seconds is comparable with the reported local memory latency of .3ms and .86 ms for the 64 processor remote memory latency. The difference arises from the addition layer introduced by MPI libraries. In spite of the difference with peak performance, the value of t_s is good compared to any other message passing architecture and hence the message-passing performance results should be comparable with the threads performance.

Since the `MPI_ALLREDUCE()` collective communication routine is used in all the message-passing algorithms presented, it is necessary to determine the communication characteristics of this routine. The interconnection topology for the SGI Origin2000 is fat bristled hypercube. The communication characteristic should be $\Theta(log\,p)$ which is also the assumption in Eq. 2.8. Numerical experiments were carried out on the SGI Origin2000. The communication time and number of processor pairs are plotted as shown in curve A in Fig 2.2.b. Curve B shows the communication time using the best curve fit of $T = k\,log(p)$. The Fig 2.2.b confirms that the assumed complexity $2(t_s + t_w)\,log\,p$ for `MPI_ALLREDUCE()` routine in Eq. 2.8. Thus it is clear that Eq. 2.8 can be used to predict the performance (T_p/iteration, S and E) of the proposed parallel algorithms.



(a) Transmission time versus number of integers transferred



(b) MPI_ALLREDUCE communication pattern

Figure 2.2: MPI communication characteristics for SGI Origin2000

2.3.2 Preconditioned conjugate gradient solver

Diagonal preconditioned conjugate gradient (PCG) performance results and scalability analysis on the SGI Origin2000 are presented in this section. The PCG algorithm is implemented using both threads and MPI. We conducted the performance study and scalability analysis with respect to one PCG iteration. Performance such as speedup (S), parallel efficiency (E) and experimentally measured serial fraction (f) are predicted using

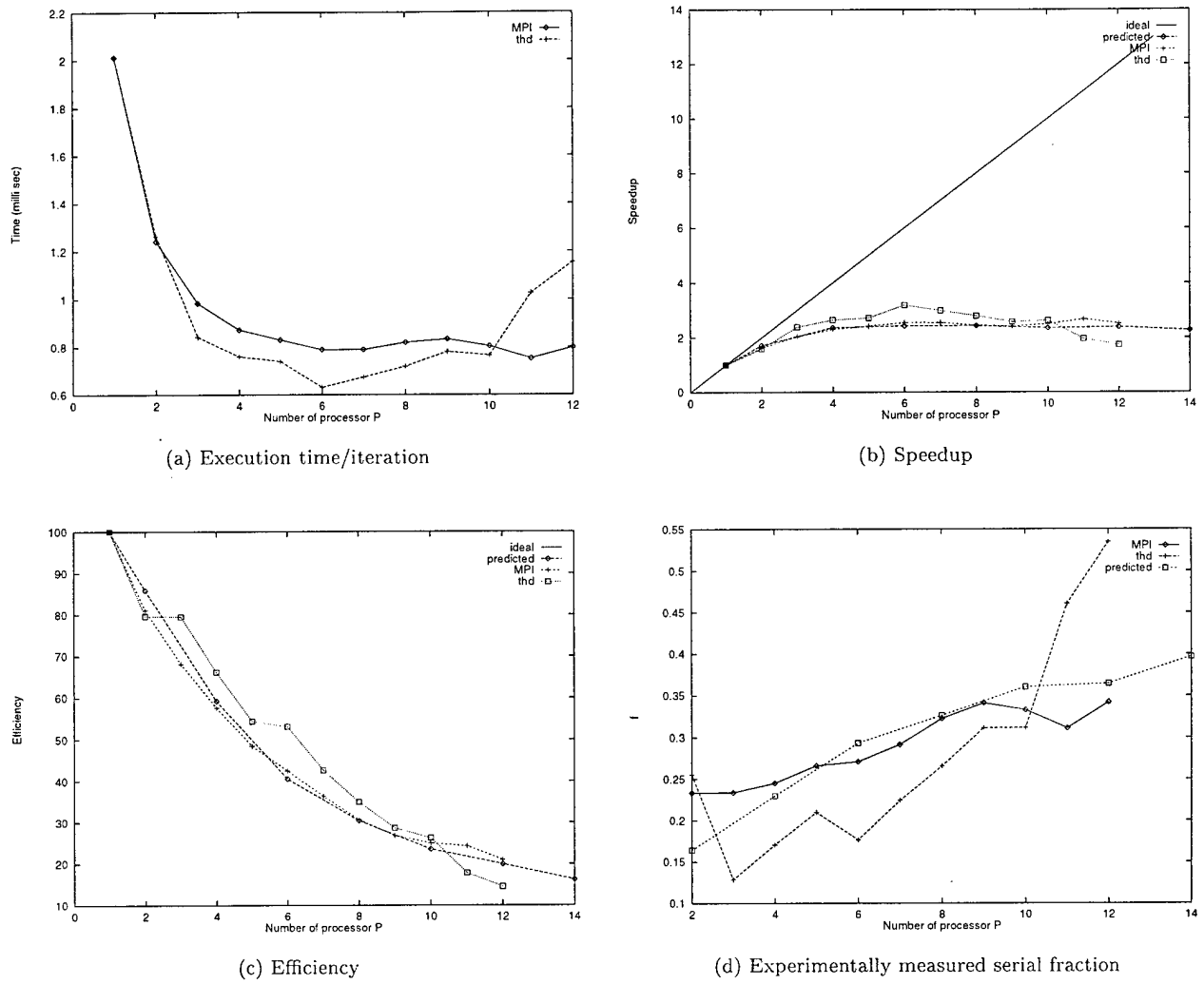
$$T_s = \underbrace{2t_c\, nnz}_{matrix-vector} + \underbrace{6(2t_c\, n)}_{vector\ operations} \tag{2.7}$$

and

$$T_p = \overbrace{2t_c\, nnz^p}^{matrix-vector} + \underbrace{\alpha\, t_s + t_w \sum_{i}^{\alpha} ibv_i}_{r,\,vector\ communication} + \overbrace{\alpha\, 2t_c \sum_{i=\alpha}^{i} ibv_i}^{r\ vector\ addition} + \underbrace{6(2t_c\, n^p)}_{vector\ operations} + \underbrace{2(t_s + t_w)\log p}_{globalsum} \tag{2.8}$$

and Eq. 2.1, 2.2 and 2.3. Where nnz and nnz^p is the number of nonzero entries in the stiffness matrix $\boldsymbol{K}$ and $\boldsymbol{K}^{(p)}$ respectively, n and n^p is the number of degrees of freedom, α is the maximum number of neighboring processors or the maximum number of adjacent sub-domains, and ibv_i is the number of interface entries of vector $\boldsymbol{p}$ information required from the neighboring processor i. The machine parameters such as t_s, t_w and t_c which are computed in previous section are used. The values of variables such as nnz^p, n^p, α and ibv_t are used from the domain decomposition analysis. With these values, Eq. 2.8 plotted along with the experimental results is as shown in Figures 2.3 – 2.6.

Figures 2.3 – 2.6 show the performance results for MESH1 to MESH5 respectively. The corresponding



(a) Execution time/iteration

(b) Speedup

(c) Efficiency

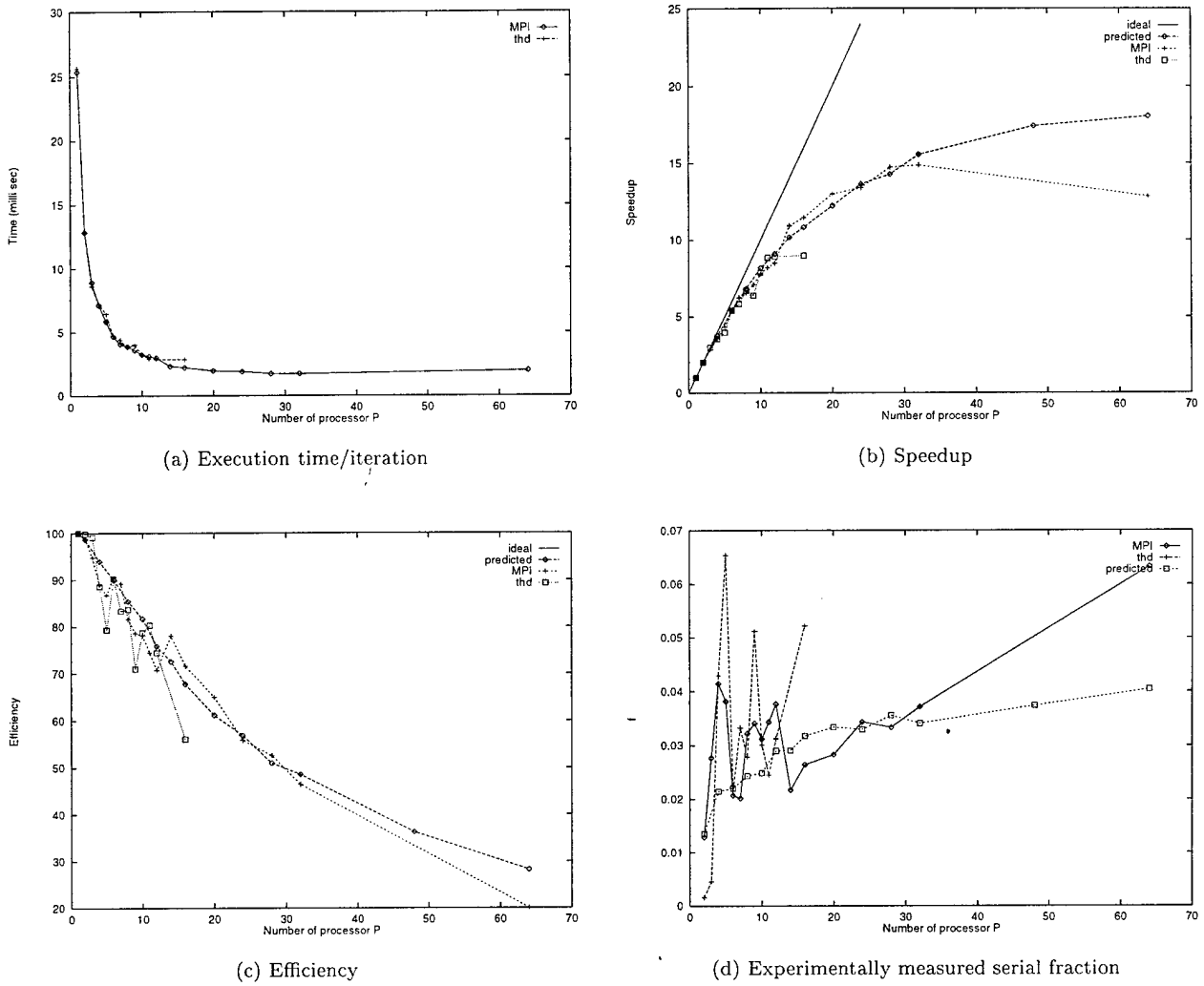
(d) Experimentally measured serial fraction

Figure 2.3: One iteration of preconditioned conjugate gradient algorithm with diagonal preconditioner for MESH1 on SGI Origin2000

tabulated values are given in Appendix C. The execution time for one iteration of the PCG is in milliseconds. The following are the inferences drawn from the performance results figures.

- Figures 2.3.a – 2.6.a of total execution time versus number of processor (p) for all the mesh sizes show that both thread and MPI implementation have almost the same performance. However for large number of processors ($p > 12$) $T_p/iteration$ for thread implementation greater than $T_p/iteration$ for MPI implementation. This can be attributed to the use of `Barrier()` construct for thread implementation algorithm versus no use `Barrier()` construct for message-passing algorithm.

- Theoretically the message-passing implementation has higher communication overhead than the threads implementation. However, for larger meshes (MESH3 and higher) S and E curves (see Fig. 2.5) shows



(a) Execution time/iteration

(b) Speedup

(c) Efficiency

(d) Experimentally measured serial fraction

Figure 2.4: One iteration of preconditioned conjugate gradient algorithm with diagonal preconditioner for MESH2 on SGI Origin2000

that MPI implementation has better performance (3% (p =5) to 13% (p =16) for both S and E for MESH3) than thread implementation.

- Thus from above two inferences it can be concluded that thread and MPI implementation perform equally well for small number of processors ($p < 12$) and small to medium mesh sizes (MESH2 and lower). However, for large number of processors ($p > 12$) and larger mesh sizes (MESH3 and higher) MPI implementation is preferred.

- As expected, Figures 2.3.b – 2.6.b of speedup versus the number of processors (p) show that speedup increases with increasing problem size. Also, for a given problem size, the speedup increases, almost linearly, with increase in the number of processors up to a certain point and then it tends to saturate.

- An almost perfect speedup till 2, 7, 14, 24 and 40 processors are achieved for MEHS1, MESH2, MESH3, MESH4 and MESH5 respectively (and then deviates and tends to saturate for a large number of processors since communication overhead dominates). The values show that a good performance is achieved when compared with analogous published findings [13, 14, 15, 16] of other researchers in related fields. Also the predicted performance results agree with the measured performance results.

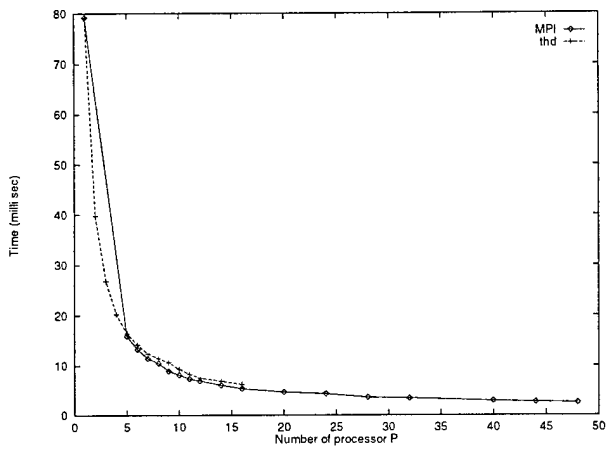
- With increase in problem size, the value of f decreases and it is slightly higher compared to the predicted values of f (see Fig. 2.3.d – 2.5.d). This can be attributed to the overhead incurred in the program such as load imbalance, machine loads and the additional layer introduced by the MPI/thread software.

- Figures 2.3.d – 2.6.d of f versus number of processor (p) shows that values of f are highly irregular and also monotonically increase with increasing p. As discussed in section 5.1.4, this is due to severe load imbalance (irregular values of f) and a monotonically increase in f is due to the communication overhead and synchronization overhead with increase in p (increase in f with increase in p).

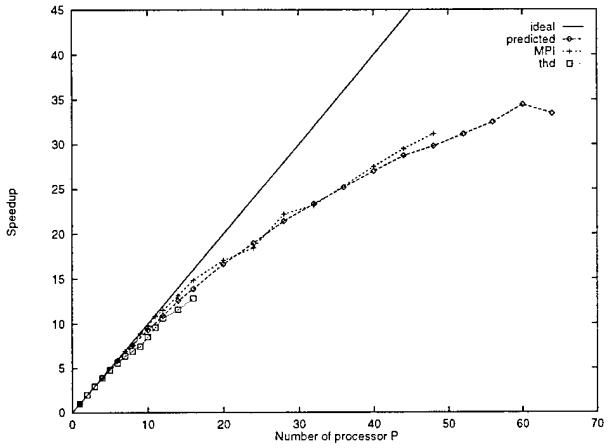
Parallel scalability analysis

Since we performed the scalability analysis with respect to one PCG iteration, the problem size W is considered to be $\Theta(n)$ where n is the number of degrees of freedom. The details of increase of problem size is given Table 2.1. We studied the rate at which n needs to grow with p for a fixed efficiency as a measure of scalability. In Figure 2.7.a, the predicted isoefficiency curves are plotted by determining the efficiencies for different values of n and p and then selecting and plotting the (n, p) pairs with nearly the same efficiencies. In Figure 2.7.b, the experimental isoefficiency curves are plotted by experimentally determining the efficiencies for different values of n and p and then selecting and plotting the (n, p) pairs with nearly the same efficiencies. Figures 2.7.a and 2.7.b show that the actual values (from the numerical experiments) agree very well with the predicted values (given by Eq. 2.8).

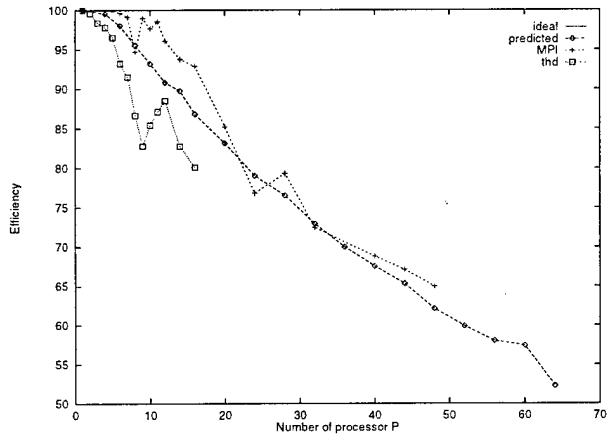
The Figure 2.7 graphically illustrates the impact of the desired efficiency on the scalability of a PCG iteration. The Figure 2.7 suggests that this a highly scalable system (combination of algorithm and parallel architecture) and requires only a linear growth of problem size with respect to p to maintain a certain efficiency. The Figures 2.7.a and 2.7.b shows that if higher and higher efficiencies are desired for a large number of processors, it becomes increasingly difficult to obtain them. The $E = 100\%$ line illustrates this fact. That is, in order to achieve an efficiency of 100% for large p ($p > 10$), the problem size W needs to be sufficiently large. An improvement in the efficiency from 70% to 80% takes little effort, but it takes substantially larger problem sizes for similar increase in efficiency from 80% to approximately 100%. As indicated in section 5.1.5, the isoefficiency curves can be used to predict the performance for a larger number of processors. If the available number of processors is equal to 64 (which is the case in this study), then the problem size W required to



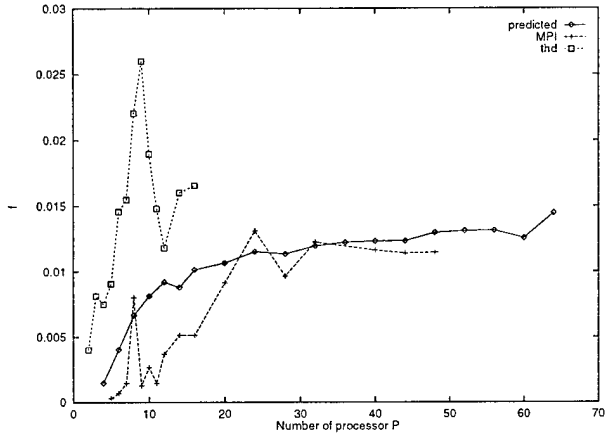
(a) Execution time/iteration



(b) Speedup



(c) Efficiency



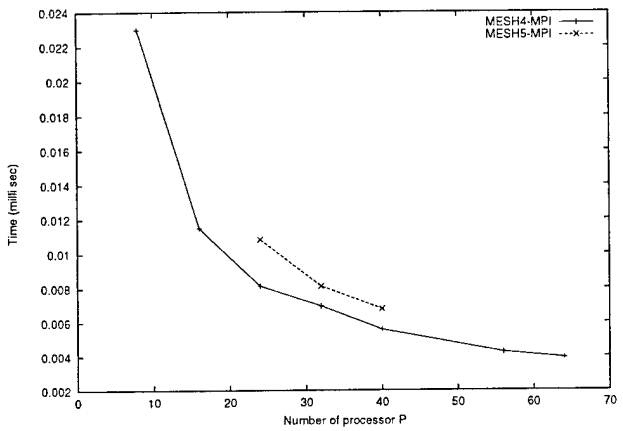
(d) Experimentally measured serial fraction

Figure 2.5: One iteration of preconditioned conjugate gradient algorithm with diagonal preconditioner for MESH3 on SGI Origin2000

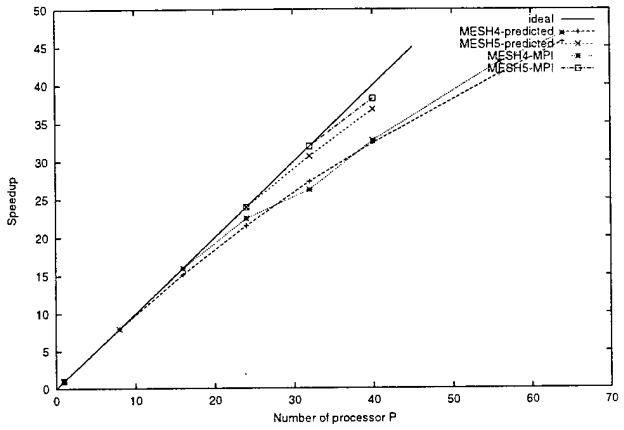
attain $E = 100\%$ will be nearly equal to 901,243. Problem sizes of $W = 659{,}412$ and $W = 413{,}549$ are required for $E = 90\%$ and $E = 80\%$ respectively.

The following conclusions can be made from the scalability analysis figures and tables.

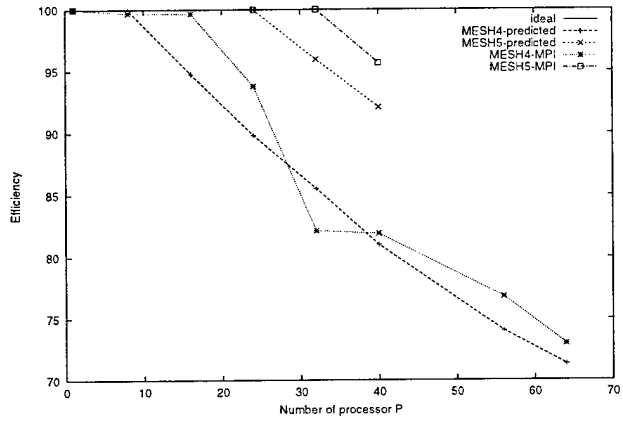
- Figure. 2.3.a and 2.3.b shows that the predicted scalability analysis agree with actual scalability analysis.
- Isoefficiency function is linear (excellent) for element based domain decomposition conjugate gradient algorithm and the SGI Origin 2000 computing platform combination.
- For MESH4 (297,576 nodes) an efficiency of 72.97% is attained with $p = 64$. This is also confirmed from the scalability analysis (ref. Table 2.3). Hence isoefficiency scalability analysis can used to



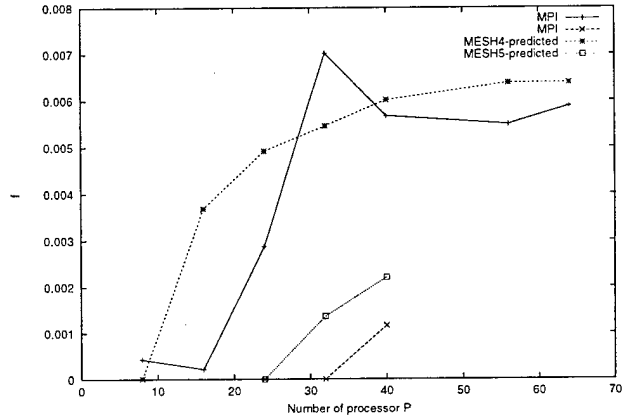
(a) Execution time/iteration



(b) Speedup



(c) Efficiency



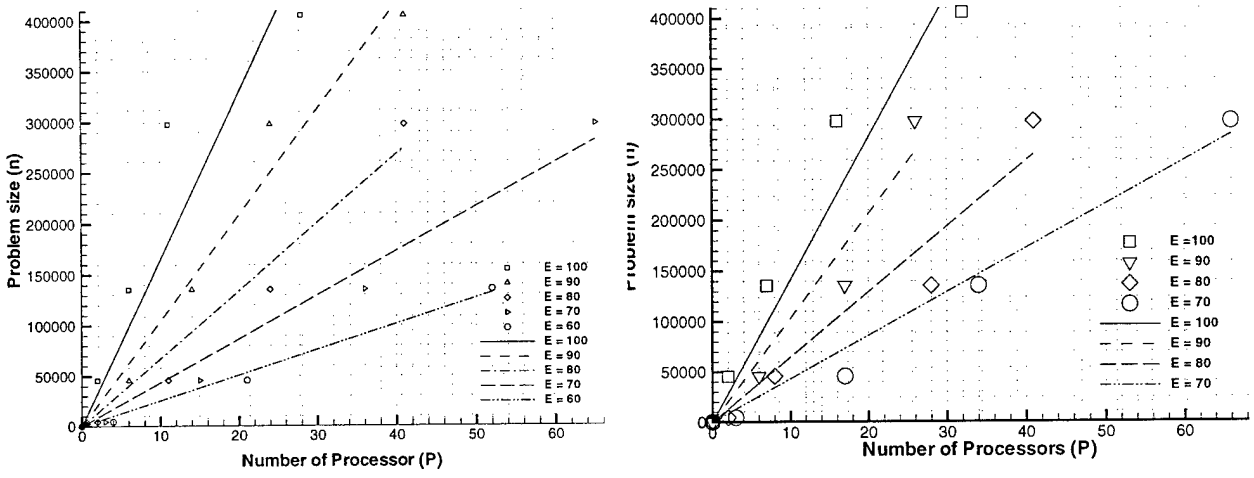
(d) Experimentally measured serial fraction

Figure 2.6: One iteration of preconditioned conjugate gradient algorithm with diagonal preconditioner for MESH4 and MESH5 on SGI Origin2000

predict accurately the problem size or the number of processors required given the number of available processors or the problem size to attained the desired efficiency for a given computing platform.

2.3.3 Risk reduction box

A complex three-dimensional structure commonly referred to as the risk reduction box geometry and manufactured by the RTM process is considered here. Since production and tooling cost are expensive for the RTM process, manufacturing simulations are necessary to avoid costly trial and error runs. From a computational perspective, the considered mold geometry with geometric complexities and multiple material regions provides a basis to study the computational validity of the parallel implementation of the explicit control volume-finite element and the implicit pure finite element mold filling methods.



(a) Predicted

(b) Actual

Figure 2.7: Isoefficiency curves for diagonal preconditioned conjugate gradient algorithm

	Predicted		Actual	
E	Slope	W =nodes	Slope	W =nodes
100%	16430.20	1,051,533	14081.93	901,243
90%	10418.00	666,752	10303.32	659,412
80%	6701.18	428,875	6461.716	413,549
70%	4329.54	277,090	4308.81	275,763
60%	2537.0	162,368	2517.21	161,101

Table 2.3: Problem size prediction for PCG for $p = 64$

A small finite element mesh consisting of 4,380 nodes and 8,670 three-noded triangular elements is considered. The mold is injected along the top edges of the risk reduction box geometry. Figures 2.8 and 2.9 shows the geometry and mesh used in the simulation. The fill contours obtained using the explicit control volume-finite element and the implicit pure finite element formulations are shown in Fig. 2.10.a and Fig. 2.10.b respectively. Isotropic permeability values of 1.0076E-6cm^2 were used in the flat regions. In the corners of the geometry, permeabilities which are 100 times higher in magnitude were used to permit race tracking. Figure 2.12 shows the performance results for RTM mold filling using the explicit control volume-finite element algorithm with/without preconditioned conjugate gradient solver on the SGI Power Challenge. Figure 2.11 shows that there is a reduction in total execution time with a diagonal preconditioner which is greater than 60% for any number of processors. The Figures. 2.11.b and Fig. 2.11.c shows that both S and E curves are almost identical for with/without diagonal preconditioner. This fact is due to the

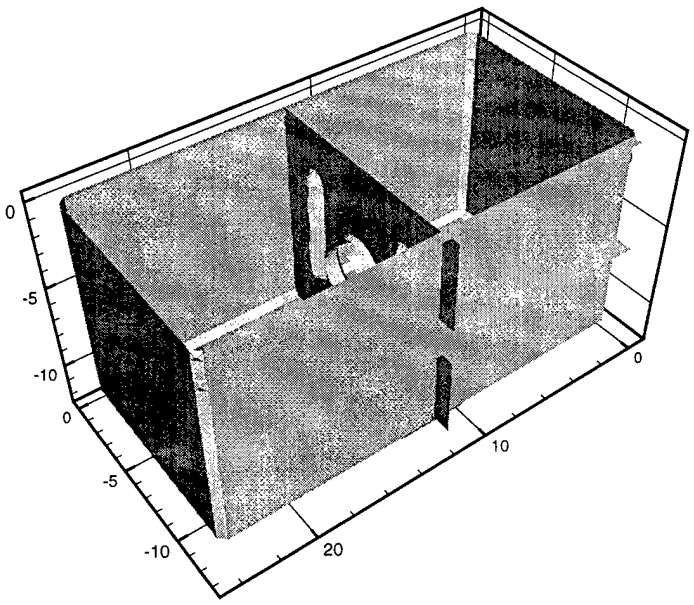


Figure 2.8: Risk reduction box geometry

only operation count difference of the preconditioner solve $M^{-1}r$ which is $\Theta(n/p)$ times more for a diagonal preconditioner. Henceforth, for the subsequent analysis, only the preconditioned conjugate gradient equation solver is considered.

Figure 2.12 shows the difference in performance between the explicit CV-FE and the implicit pure FE mold filling approaches on the SGI Power Challenge and the SGI Origin2000. Earlier studies [6] concluded that computational cost-wise isothermal RTM mold filling simulation with the implicit pure finite element algorithm performs better than the explicit control volume-finite element algorithm as the former does not have restrictions on the time step size chosen. For one processor runs, the total execution time figures confirms this fact. A time step of Δt=1.0 sec was employed for the implicit pure finite element scheme which is sufficient to achieve good resolution of the total fill time. Total execution time curves shows that the SGI Origin2000 is a better parallel architecture than the SGI Power Challenge. The pure FE algorithm has lower execution time than the explicit CV-FE algorithm for any number of processors. However, in terms of S, E and f, the explicit CV-FE performs better than the pure FE approach on the SGI Power Challenge and the pure FE algorithm performs better than explicit CV-FE approach on the SGI Origin2000 (architectural issues). Figure 2.12.d shows that there is a monotonic increase in the value of f. As discussed in section 5.1.4, this indicates that the loss of efficiency is only due to the communication overhead. Therefore, loss in performance due load imbalances are negligible.

This section validates the parallel implementation of explicit CV-FE and implicit Pure FE approaches on both the SGI Power Challenge and the SGI Origin 2000 computing platforms. Also, it can be concluded

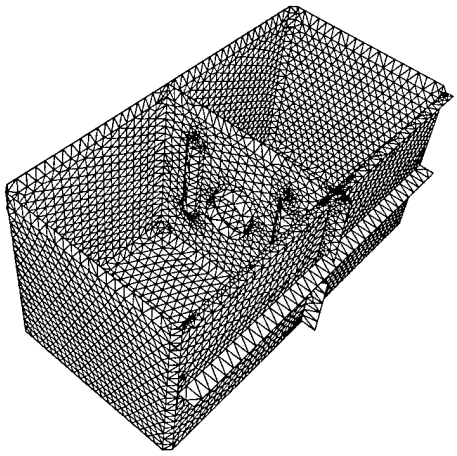


Figure 2.9: Finite element mesh of risk reduction box, 4,380 nodes and 8,670 elemen ts, MESH1

that diagonally preconditioned conjugate gradient solver can be effectively used for RTM molding filling simulations.

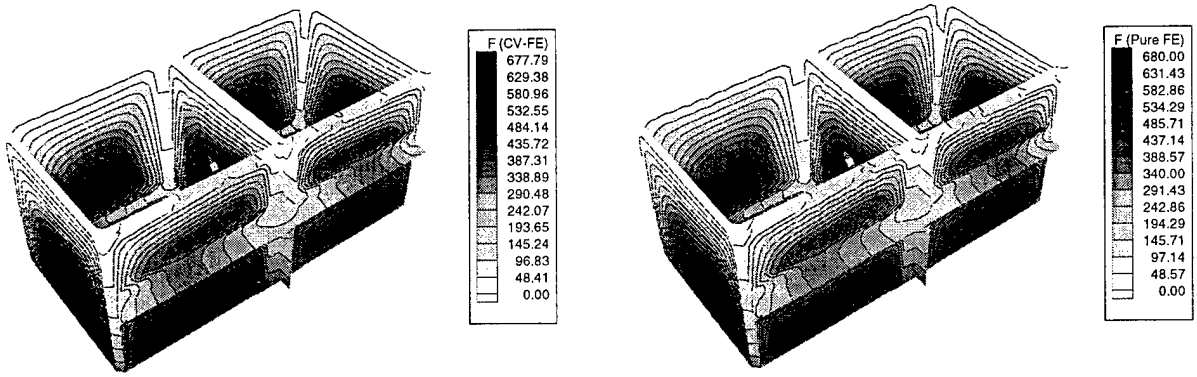
2.3.4 Commanche keel beam

The previous example shows that an accurate parallel analysis can be performed using the methods under study. However, due to the very small number of elements involved, it is not a good test of the efficiency of the program. Hence, larger problem sizes were investigated.

A geometrically complex large scale composite structure of the 24 feet keel beam is considered for the performance and scalability studies. The mesh geometries for thin section process simulations involved two-dimensional triangular elements built in a three-dimensional space. The 24 feet keel beam geometry is shown in Fig. 1.2. The details of four different mesh configurations (MESH2, MESH3, MESH4 and MESH5) employed are shown in Table 2.1.

MESH2

Figure 2.13 shows the performance results with the explicit CV-FE approach on the SGI power challenge and the SGI Origin2000. In terms of S, E and f, Fig. 2.13 shows the good performance and an approximately close to perfect speedup till 10 processors. However, the CV-FE algorithm with one processor run took 126.2 hours on the SGI Power Challenge and 80.6 hours on the SGI Origin2000. The execution time was lowered to 4.2 hours using 32 processors on the SGI Origin2000. The mesh size ratio between MESH2 and MESH1 is $\approx$30. The total execution time wise ratio between MESH2 and MESH1 is $\approx$137 for the SGI power challenge and $\approx$190 for the SGI Origin2000. This clearly suggests that total execution time for the CV-FE algorithm for moderate size and higher size meshes are not within the *Reasonable Time*. In contrast the pure FE approach took 44.2 minutes with 1 processor and 4.2 minutes with 64 processors on the SGI



(a) CV-FE formulations

(b) Pure FE formulations

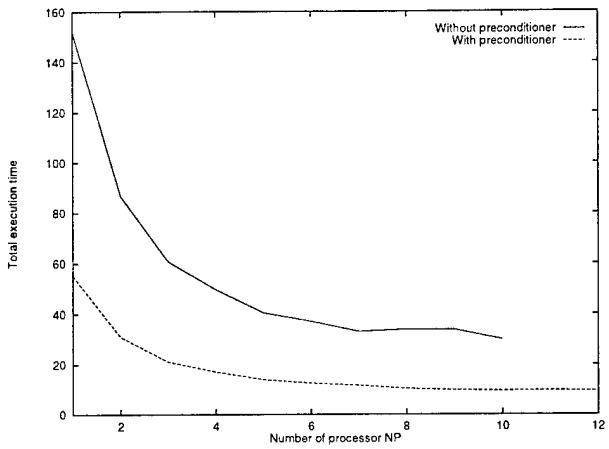
Figure 2.10: RTM mold fill contours

Origin2000 which clearly shows the computational speedup of using the pure FE approach over the CV-FE approach. Henceforth, from practical considerations only the implicit pure FE approach was considered for subsequent large scale problem analysis. Figure 2.13 shows the performance results with the implicit pure FE algorithm on the SGI Origin2000. A time step size of 5.0 seconds was employed here. Figures 2.13.d and 2.14.d show that the values of f are highly irregular. As discussed in section 5.1.4, this indicates that there exists severe load imbalance and hence the loss in parallel performance. Also, it indicates that the loss of parallel performance due to communication overhead is small. The choice of good domain decomposition methods which preserve the flow front physics will reduce the load imbalance.

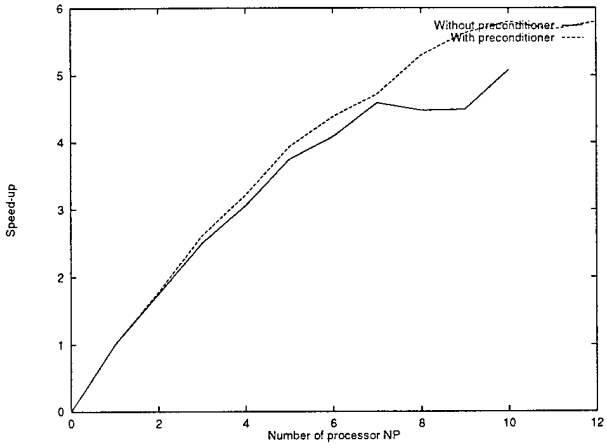
MESH3

Figures 2.18 and 2.19 show the performance results for MESH3 with the implicit pure finite element mold filling scheme. Here, a time step size of Δt =5.0 sec for the SGI Power Challenge and Δt =0.2, 1.3, 2.6, 5.0 and 5.2 sec on the SGI Origin2000 were used. A maximum of 64 processors were employed for the study. As expected, it is clear from the previous examples (MESH1 and MESH2) and the present example (MESH3) that as the mesh size is increased, there is an increase in performance in terms of both speedup and parallel efficiency.

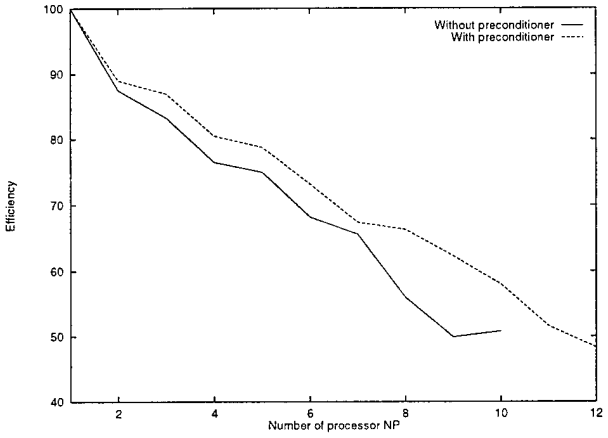
Figure 2.18.e shows the PCG iteration count versus the percentage of mold cavity filled. The iteration count increases as the percentage of mold cavity is filled. This also justifies the use of the conjugate gradient iterative linear equation solver for the mold filling applications as opposed to the use of a direct solver. Figures 2.18.e and 2.18.f show that the conjugate gradient method takes very few iterations in the initial stages and it increases gradually. Thus the iterative solver takes less time in the beginning and gradually increasing as opposed to the direct solver which takes a constant time for all mold filling iterations, thereby,



(a) Total execution time



(b) Speedup

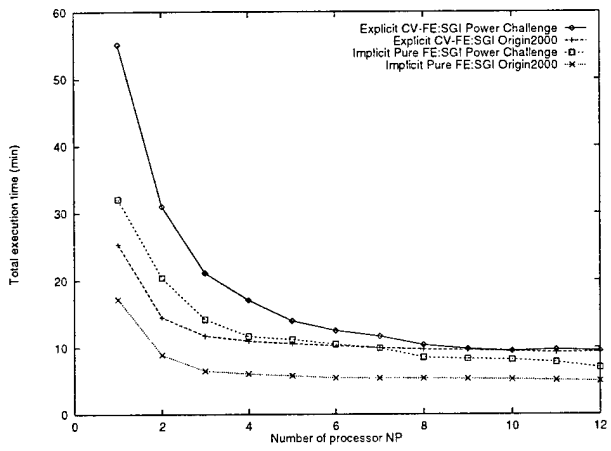


(c) Efficiency

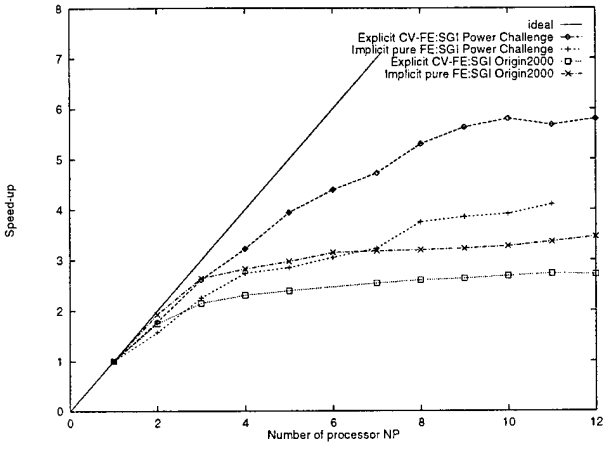
Figure 2.11: Explicit RTM mold filling algorithm and with/without preconditioned conjugate gradient solver for MESH1 and SGI Power Challenge

increasing the overall execution time.

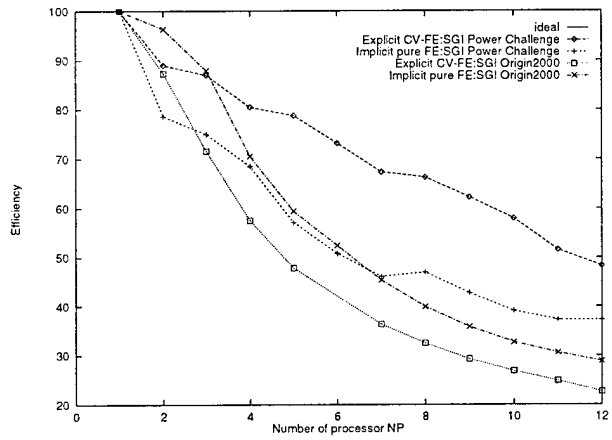
Figure 2.18.a shows that the total execution time (T_p) decreases with increase in time step size. The reduction of the total execution time is due to the decrease in the total number of mold filling iterations with increase in time step size, Figure 2.18.b. The horizontal portion of the curves of Fig 2.18.b suggests that the implicit Pure FE mold filling algorithm tries to balance or conserve the resin mass and hence the increase in percentage of mold cavity being filled is very small. Therefore it can be said that the fluid flow front tracking algorithm is effectively inactive during horizontal portion of the curve. Hence, as the time step size is increased, the total number of mold filling iterations with the active front tracking algorithm increases and hence the increase in load imbalance between the processors consequently the decrease in performance S and E. Figure 2.18.b shows that for $\Delta t = 0.2$, the total number of mold filling iterations with the front



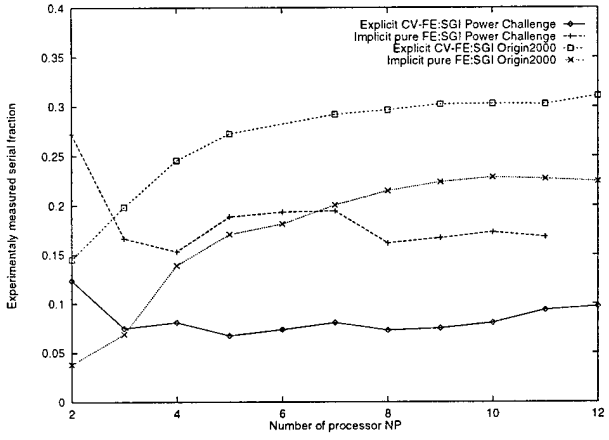
(a) Total execution time



(b) Speedup



(c) Efficiency

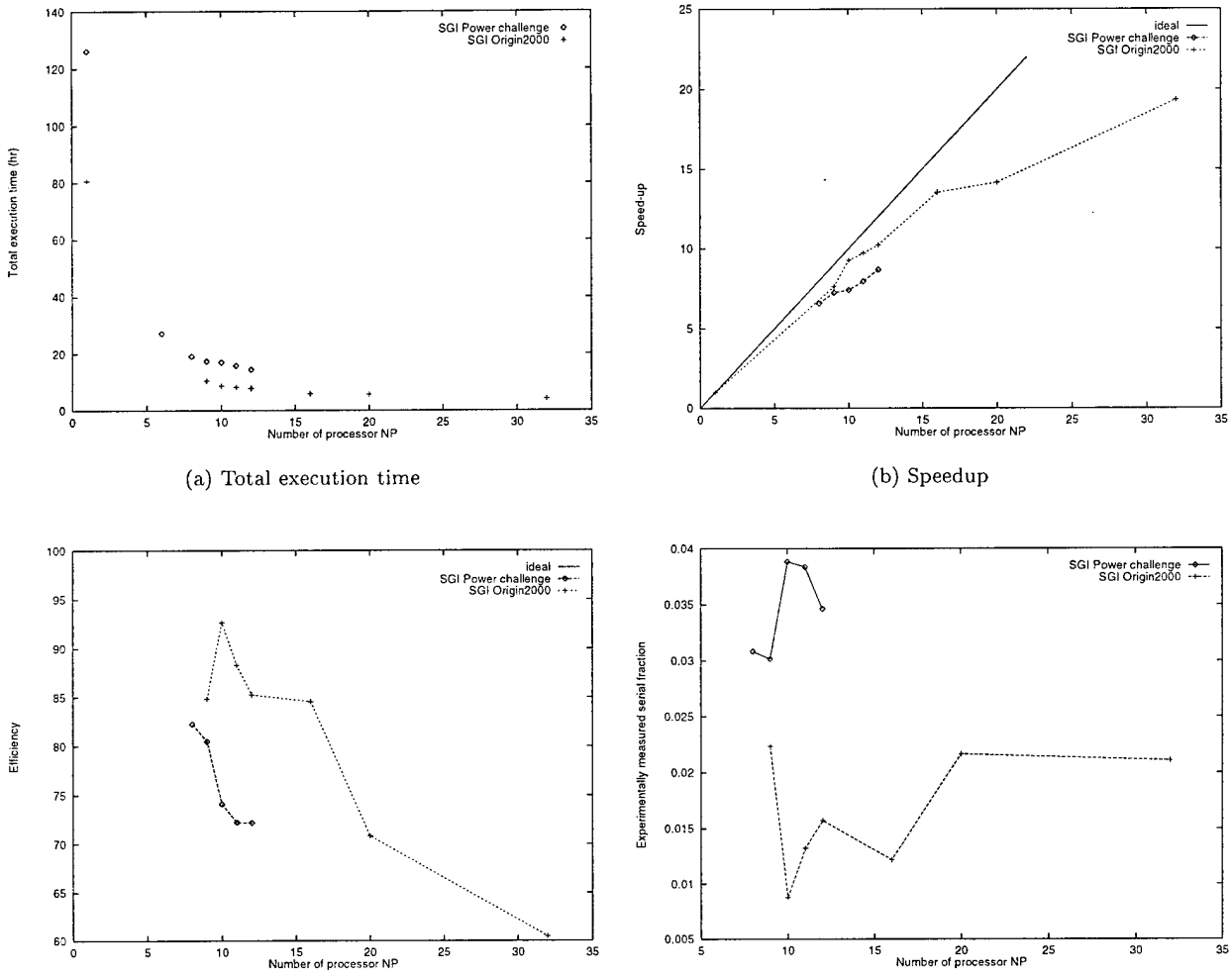


(d) Experimentally measured serial fraction

Figure 2.12: RTM mold filling with CV-FE and pure FE algorithm MESH1 on SGI Power Challenge

tracking inactive is more compared to the other time step sizes. This fact is also illustrated in Fig 2.19.d, where the value of f is highly irregular for Δt =5.0 sec and the value of f is fairly constant for Δt =0.2 sec. Hence optimal performance (T_p, S and E) with sufficient flow front resolution is dependent on optimal selection of time step size Δt.

Figures 2.18.c and 2.18.d show the number of sub-domains being filled with the percentage of mold filling. The pattern varies with the number of sub-domains (number of processor) and the time step size. Once a sub-domain is filled there is no need of front tracking in that sub-domain. Therefore, this is another cause of load imbalance. Ideally, there should be a steady increase in the number of sub-domains being filled from the start of the the mold filling run. Figure 2.18.c shows that after 70% of the mold fill, the sub-domains start being filled for p =7,9,10 and Fig 2.18.d shows that after 30% of mold fill, the sub-domains starts being filled for p =44. This is also reflected in Fig. 2.19.d where for small p ($p < 10$), the values of f



(a) Total execution time

(b) Speedup

(c) Efficiency

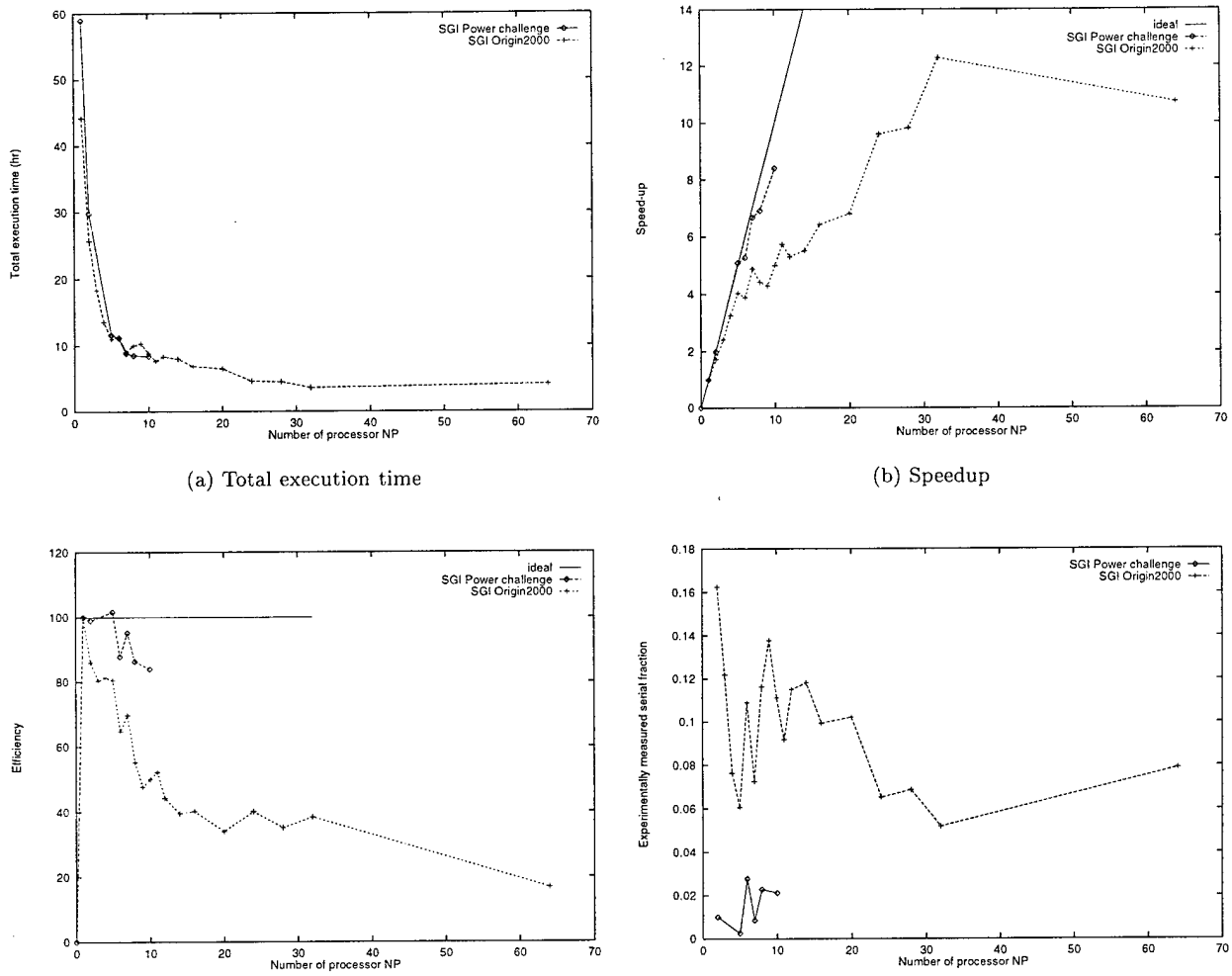
(d) Experimentally measured serial fraction

Figure 2.13: RTM mold filling using explicit mold filling algorithm and preconditioned conjugate gradient solver for MESH2 on SGI Power Challenge and SGI Origin2000

are highly irregular and for larger p ($p > 10$), the values of f is fairly constant. Thus parallel performance depends on sub-domain filling pattern which greatly depends on how the sub-domains are created (domain decomposition) and the flow front pattern in the mold (position of injection ports and time step size) with respect to the sub-domains.

MESH4, MESH5

Larger finite element meshes with 297,576 nodes and 594,756 triangular elements (MESH4) and 405,327 nodes and 809,505 elements (MESH5) are considered next. The performance results along with the performance results of the other meshes are shown in Fig 2.20. The figure clearly shows that as the mesh size is increased, the performance is also increased. A one processor run took about 108 hours and 187.68 hours on MESH4



(a) Total execution time

(b) Speedup

(c) Efficiency

(d) Experimentally measured serial fraction

Figure 2.14: RTM mold filling using implicit mold filling algorithm and preconditioned conjugate gradient solver for MESH2 on SGI Power Challenge and SGI Origin2000

and MESH5 respectively. A 64 processor (Maximum available) run took 2.28 hours for MESH4 and a 40 processor run took 4.72 hours for MESH5. An almost linear speedup of 24 and 40 is achieved for MESH4 and MESH5 respectively.

2.3.5 Parallel scalability analysis of pure finite element mold filling scheme

When the parallel scalability of the whole scheme (which also includes linear solver) is considered, more consideration has to be given to the selection of problem size W. The choice can be either the number of elements or the number of nodes. However, in the present study, the ratio of the number of elements to the number of nodes for the mesh sizes (MESH1 to MESH5) is nearly constant ($\approx$2). Therefore, either the number of elements or the number of nodes can be chosen as the problem size W. It is also important to note

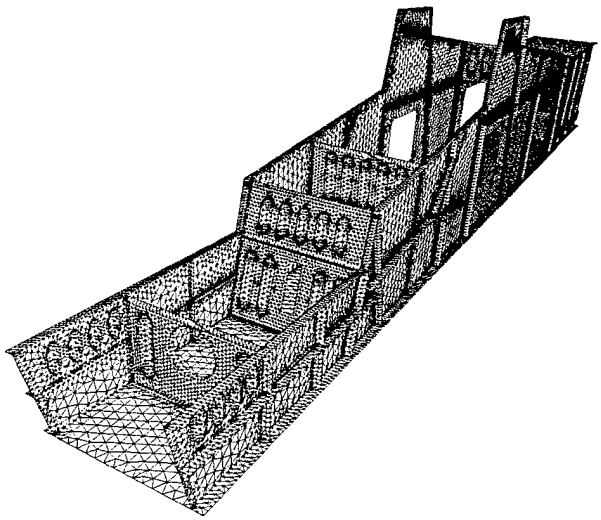


Figure 2.15: Finite element mesh of keel beam, 45,547 nodes and 89,945 elements

that the present results of scalability analysis are only valid for number of elements to the number of nodes ratio being approximately equal to two. For any other ratios the scalability analyses have to be repeated.

E	Number of elements	Number of nodes	Slope
100%	1,774,462	887,231	27725.97
90%	1,345,570	672,785	21024.53
80%	945,064	472,532	14766.63
70%	569,493	284,747	8898.33

Table 2.4: Mesh size prediction for $p = 64$

In Figure 2.21 the experimental isoefficiency curves are plotted by experimentally determining the efficiency for different values of n (number of elements) and p and then selecting and plotting the (n, p) pairs with nearly the same efficiencies. There exists small inconsistencies in the plot which can be attributed to different time step sizes used and variation of boundary conditions from mesh to mesh. The effect of time step size on the parallel performance is discussed in section 5.2.4.2.

Figure 2.21 suggests that the parallel finite element system (combination of parallel finite element algorithm and SGI Origin200 parallel architecture) is a highly scalable system since it only requires a linear growth of problem size with respect to p to maintain a certain efficiency. If the available number of processors is equal to 64 (which is the case in this study) then the problem size W (number of elements) required to attain $E =$ 100% will be nearly equal to 1,774,462 elements (887,231 nodes). For other efficiencies the required problem size is tabulated in Table. 2.4. For MESH4 (594,756 elements and 297,576 nodes) an efficiency of 73.9% is attained with $p = 64$. This is also confirmed from the scalability analysis (ref. Table 2.4).

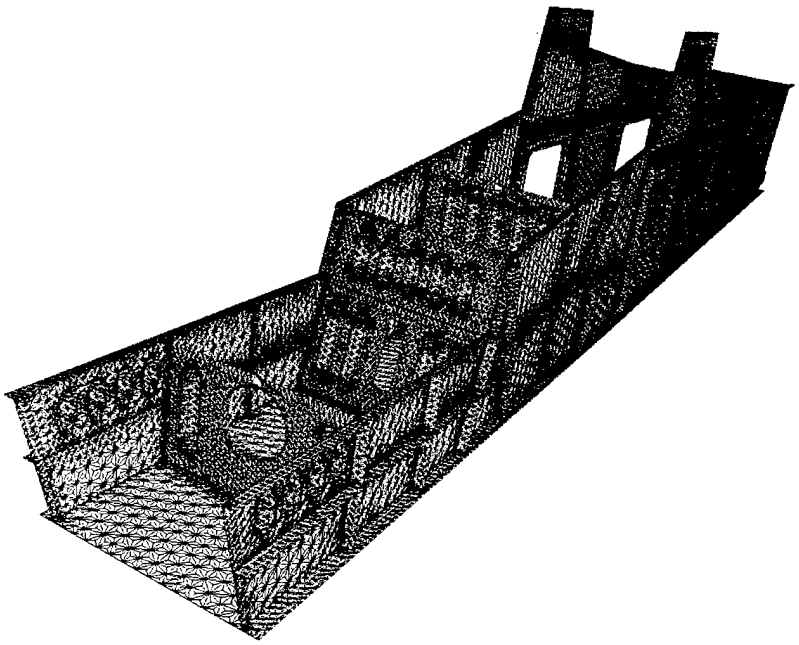


Figure 2.16: Finite element mesh of keel beam, 135,492 Nodes and 269,835 elements

2.4 Closure

Parallel performance results are reported for the traditional explicit CV-FE mold filling scheme and the implicit pure finite element mold filling scheme via an element based domain decomposition with a diagonal preconditioned conjugate gradient linear equation solver (PCG). For the PCG algorithm, 1.77 Giga-flops performance is achieved with 64 processors and mesh with 297,576 degrees of freedom. The performance results are excellent when compared to other relevant results published in the literature for parallel finite element analysis. The finite element meshes considered and the results obtained illustrates the practicable applicability of the parallel finite element formulations. Performance results of the thread model implementation is almost the same as that of the MPI model implementation. Henceforth, for maximum portability of the program, MPI implementation is the better choice of the two.

Scalability analysis showed that the implicit pure finite element method is highly scalable and requires only a linear growth of problem size with respect to the number of processors to maintain a certain efficiency. The scalability analysis suggests that to achieve 100% efficiency, the required mesh size needs to be nearly equal to 1,774,462 elements and 887,231 nodes and for 90% efficiency a mesh size of 1,345,570 elements 672,785 nodes is required. Also, the scalability analysis showed (see Table. 2.3) that the parallel diagonally preconditioned conjugate gradient (PCG) parallel system (PCG algorithm with SGI Origin2000 parallel architecture) is highly scalable and requires only a linear growth of the problem size with respect to the number of processors to maintain a certain efficiency. The scalability analysis (see Table. 2.4) of the PCG suggests that to achieve 100% efficiency the required mesh size needs to be nearly equal to 901,243 nodes,

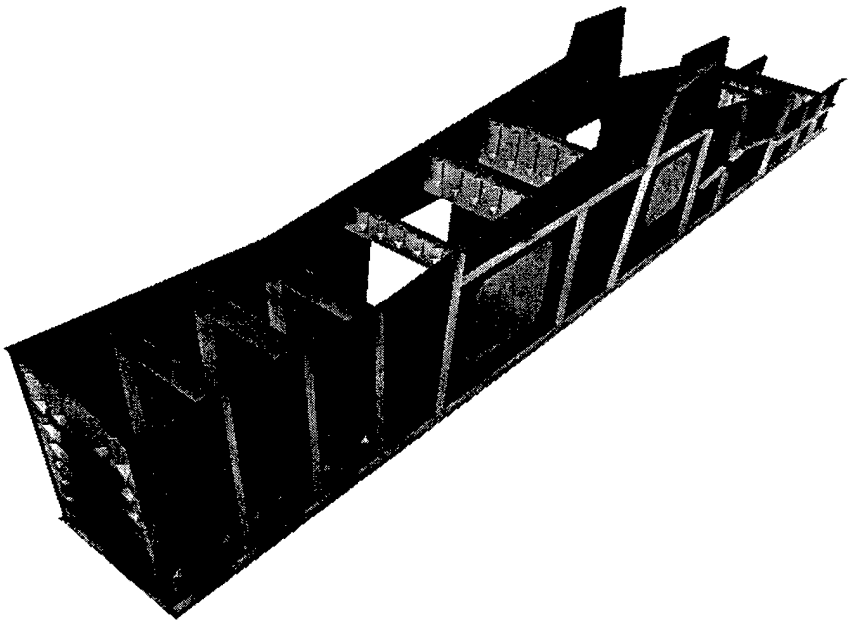
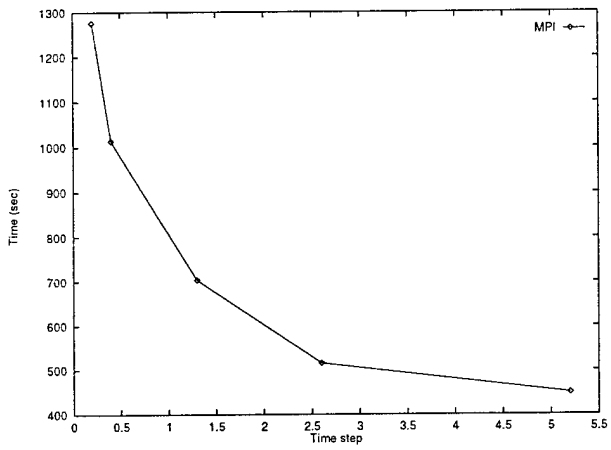


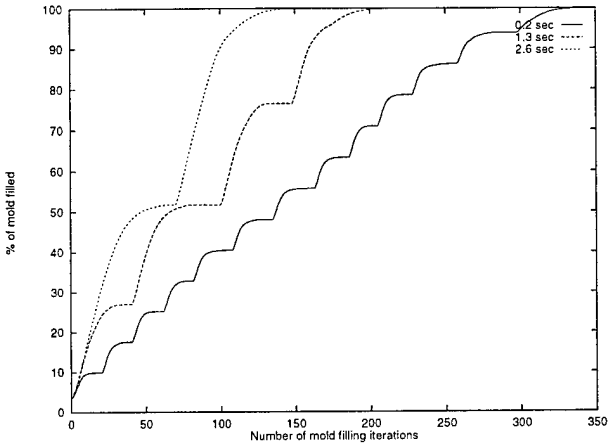
Figure 2.17: Fill contours for 24 feet keel

and for 90% efficiency, a mesh size of 659,412 nodes is required.

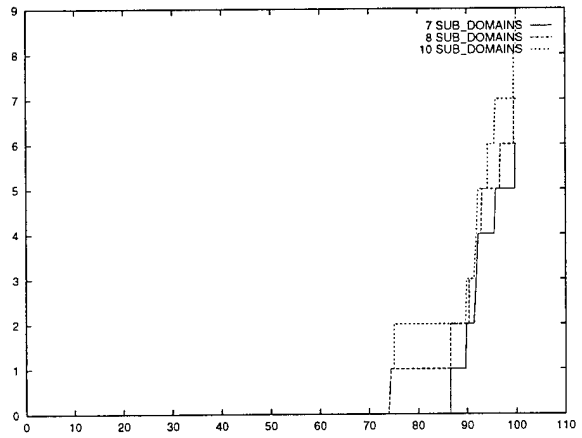
On an SMP running a full featured operating system, the operating system and its demons (demons include network demons, file system, accounting, message logging etc.) can consume enough CPU resources to cause performance problems for applications which use static load balancing schemes. It is observed that on an SMP with p processors (64 in the case SGI Orgin2000), the application user may want to use only $p-1$ processors, leaving the last for the operating system.



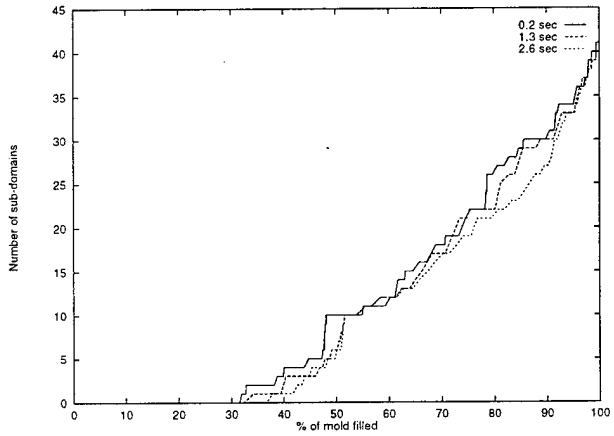
(a) Total execution time for different time steps with p=44



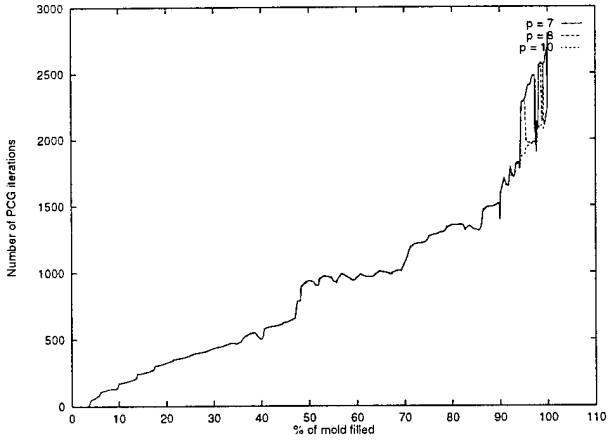
(b) percentage of mold fill with mold filling iterations p=44



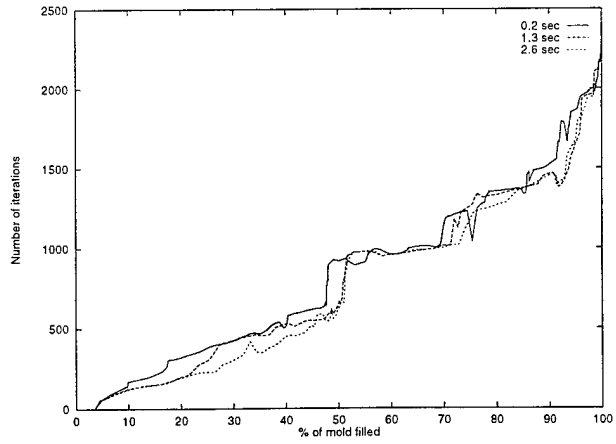
(c) Sub-domain filling pattern for p=7,8,10 and Δt =0.2 sec



(d) Sub-domain filling pattern for p=44

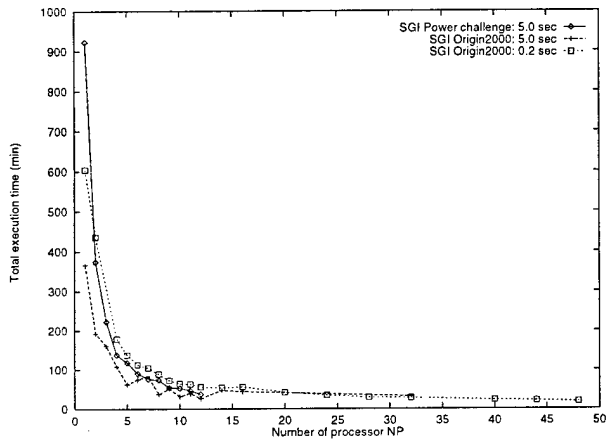


(e) Preconditioned conjugate gradient iteration count p = ,7,8,10

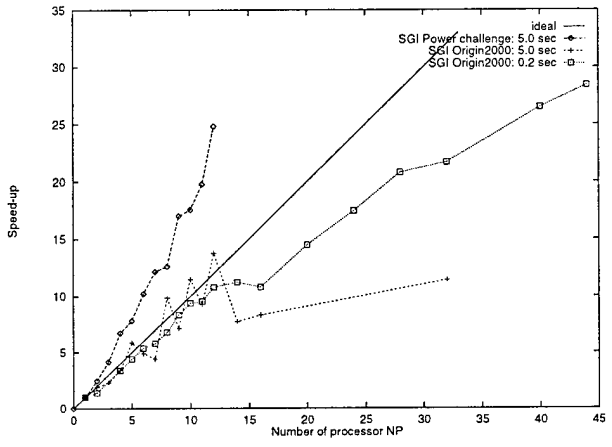


(f) Preconditioned conjugate gradient iteration count p = 44

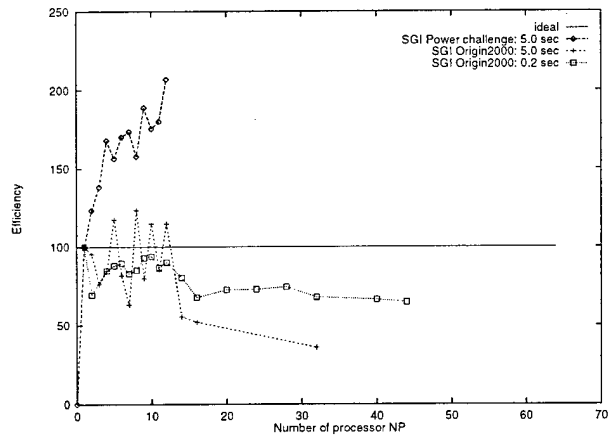
Figure 2.18: Pure FE mold filling approach for MESH3



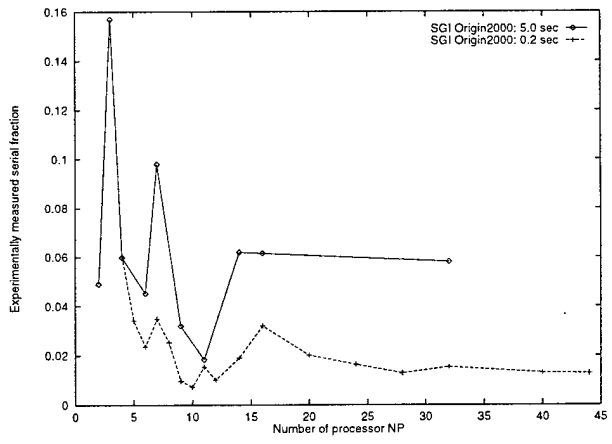
(a) Total execution time



(b) Speedup

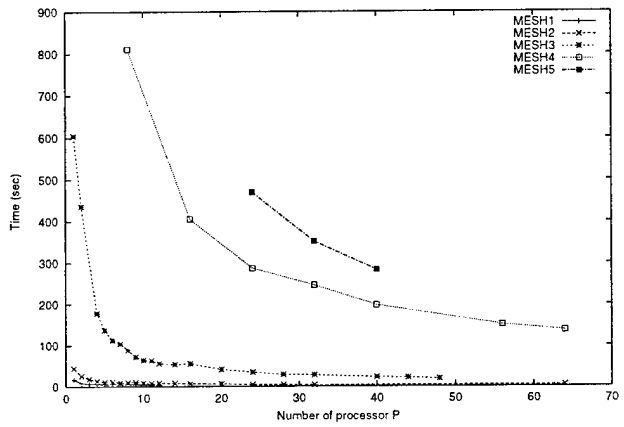


(c) Efficiency

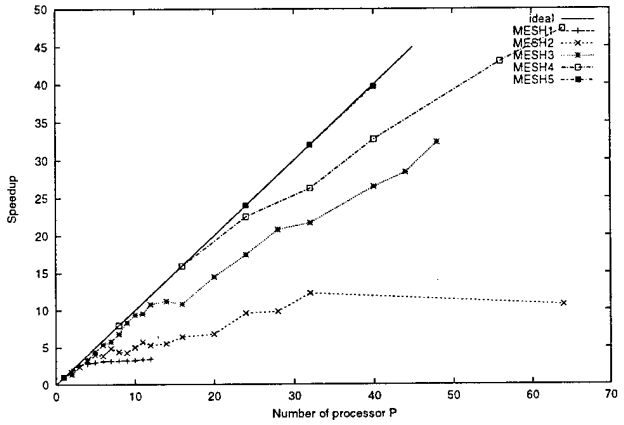


(d) Experimentally measured serial fraction

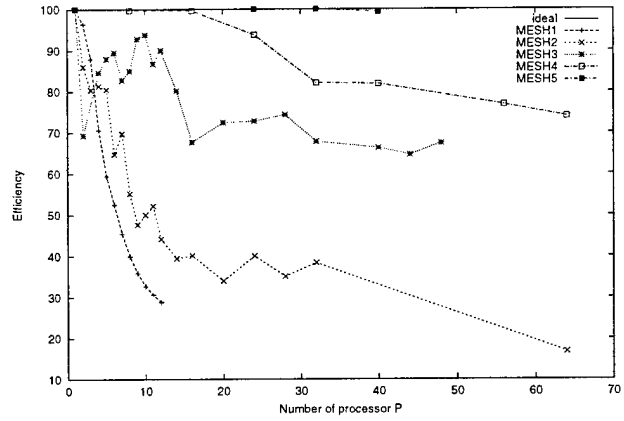
Figure 2.19: RTM mold filling with pure FE approach and a preconditioned conjugate gradient algorithm with diagonal preconditioner for MESH3 on SGI Origin2000.



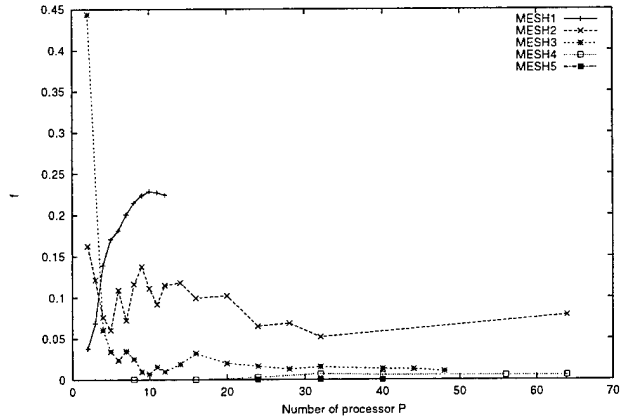
(a) Total execution time



(b) Speedup



(c) Efficiency



(d) Experimentally measured serial fraction

Figure 2.20: RTM mold filling with pure FE approach and the preconditioned conjugate gradient algorithm with diagonal preconditioner for MESH1-MESH5 on SGI Origin2000.

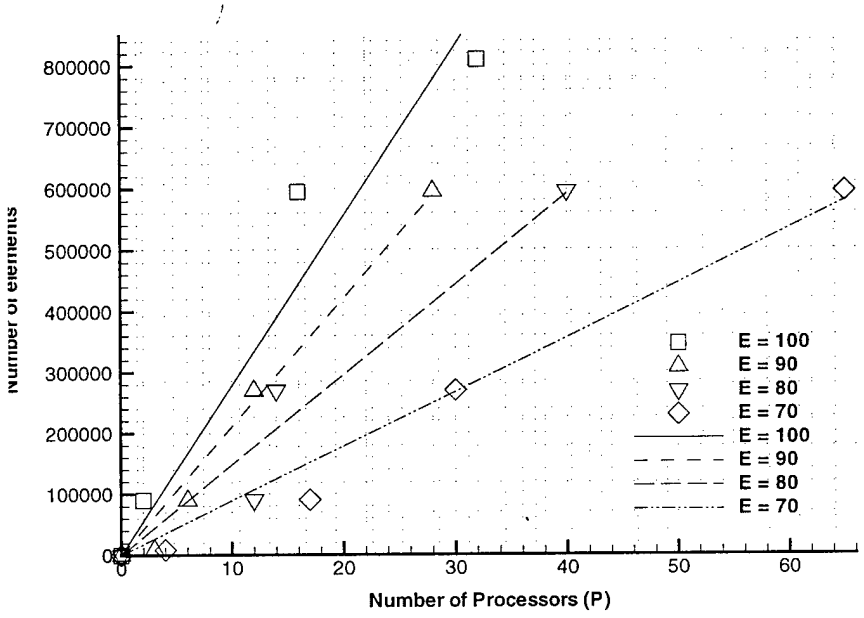


Figure 2.21: Isoefficiency curves for the pure finite element mold filling approach on the SGI Origin2000.

Concluding Remarks and Future Directions

3.1 Introduction

The objectives of the present efforts were to implement and investigate the computational modeling and simulation techniques and developments encompassing: (a) parallel computations on symmetric multiprocessors (SMP) and its performance and important issues related to these topics, (b) computational algorithms for providing general parallel finite element modeling analysis and important issues such as performance and portability, and (c) implementation of parallel computational algorithms for mold filling techniques through the use of finite element analysis. Specifically, free surface flow problems in resin transfer molding that occur during resin impregnation of fiber preforms inside complex mold cavities which are characterized by the pressure driven flow fields have been investigated. At this juncture, the following conclusions are drawn:

3.1.1 Parallel computations on symmetric multiprocessors (SMP)

- Current generation SMP's with medium to large number of processors can be effectively used as a supercomputer (Massively Parallel Platforms, MPP) for large scale complex geometries in scientific and engineering problems.
- Message-passing programming models perform almost equal to that of thread based programming models.
- As combined architecture model is rapidly becoming available and also future research appears to be heading in this direction, it is necessary to develop parallel programming models for the same. The following programming models can be immediately envisioned for the combined architecture model:

1 **Hybrid thread and message-passing model:** With n processes over s SMP's with threads communicating via shared-address space within SMP and message-passing between SMP through sockets.

2 **Message-passing MPI model:** MPI model with n process over s SMP's with shared memory message-passing within a SMP and message-passing between SMP through sockets. In the FEM context, partition the mesh into several sub-meshes equal number of SMP's (s) and then partition each sub-meshes further to equal to the number of available processors (p) in each SMP.

3 **Hierarchical MPI communication model:** In this case, tasks within a SMP may be divided into several groups, with shared-memory communication between tasks within a group, and sockets between groups within the same SMP. Tasks between SMP communicate via sockets. In the FEM context, partition the mesh into several sub-meshes equal to the total number of available processors ($s \times p$).

Via this study, option 1 is eliminated as thread model implementation results are almost same as of MPI implementation. This makes option 2 and 3 to be investigated further.

3.1.2 Computational algorithms for providing general parallel finite element applications

- Mesh partitioning with message-passing algorithms provide coarse grain parallelism with high degree of concurrency and data locality. It also provides portable programs for a wide range of parallel architectures without compromising performance.
- With the above mentioned frame work and with the proper selection of algorithms for solving sub-problems (linear solver, stiffness assembly, etc.) inside the finite element analysis and with element based mesh partitioning schemes, minimum modification is required for the 'old' serial codes to be used in parallel architectures without compromising the performance.
- Experimentally measured serial fraction characteristics can be an effective diagnostic tool to find out causes for performance loss in the parallel algorithm implementation.
- Parallel scalability analysis in terms of isoefficiency function is an effective tool for parallel computation performance evaluation. It can be effectively used to predict the mesh size or number of processors required to obtain certain efficiency from parallel computing systems, given the number of available processors or mesh size apriori respectively.

3.1.3 Parallel computational algorithms for Resin Transfer Molding

- Implicit pure finite element mold filling algorithm is computationally superior compared to the explicit control-volume finite element algorithm (CV-FE) on different computational platforms (SGI Power

Challenge and SGI Origin2000). For large scale problems CV-FE algorithms become impossible to analyze within *Reasonable Time* or realistically impossible even with parallel processing for large scale problems.

- CV-FE algorithms have better load balance among the parallel processors than the implicit pure FE algorithm as in the former case only few control volumes gets filled in each mold filling iteration.
- The overall effectiveness of both the computational methodologies for practical applicability to complex composite structure was demonstrated.

3.1.4 Recommended areas for future Directions

- Investigate the proposed parallel methodology on other SMP's with different interconnection network and also on message-passing parallel architecture such as cluster of workstations over Ethernet and cluster of SMP's.
- Investigation of element based domain decomposition conjugate gradient algorithm with different advanced preconditioners such as incomplete Cholesky factorization.
- Investigation of other existing conjugate gradient like iterative linear solvers.
- Investigation of direct linear equation solvers such as the Schur complement method with iterative solver for interface solution and Cholesky factorization method with nested dissection type reordering.
- Investigation of multi-disciplinary (flow, thermal, and structural interactions) parallel finite element analysis on SMP's.

3.2 Acknowledgments

Special thanks are particularly due to Dr. Andrew Mark and Mr. William Mermagen Sr. of the CICC Center and the Integrated Modeling and Testing (IMT) Technical Activity, Army Research Laboratory, APG, MD, for their continued encouragement and support. The computational support and enthusiasm of Dr. C. Nietubicz at the ARL MSRC center and additional support in the form of computer grants from Minnesota Supercomputer Institute (MSI) are also gratefully acknowledged. Additional thanks are also due to Dr. Ram Mohan and Mr. Dale Shires for the support by providing the finite element meshes for the study.

Bibliography

[1] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing/ Design and analysis of Algorithms.* The Benjamin/Cummings Publishing Company, Inc., 1994.

[2] J. M. Castro, B. F. Smith, and S. I. Guceri. Experimental and numerical analysis of resin impregnation during the manufacturing of composite materials. In *Proc. Amer. Soc. Comp.,2nd Tech. Conf.*, pages 209–217, Larcasta, 1987.

[3] M. V. Bruschke and S. G. Advani. A finite element/control volume approach to mold filling in anisotropic porous media. *Poly. Comp*, 11(6), 1990.

[4] R. V. Mohan, N. D. Ngo, K. K. Tamma, and K. D. Fickie. Three dimensional developments for thick RTM composites. *J. Poly. Sci. Eng.*, 1995. submitted.

[5] R. V. Mohan, N. D. Ngo, K. K. Tamma, and K. D. Fickie. On a pure finite element based methodology for RTM simulations. *ARL-TR-975*, March 1996.

[6] K. K. Tamma et al. An overview of recent developments and advances in resin transfer molding of thin/thick composite structures. *ARO Report*, October 1994.

[7] V. R. Voller and S. Peng. An algorithm for analysis of polymer filling of molds. *Polymer Eng. Sci.*, 35(22):1758–1765, 1995.

[8] F. Trochu, R. Gauvin, and D. M. Gao. Numerical analysis of the resin transfer molding process by finite element method. *Advances in Polymer Tech.*, 12(4):329–342, 1993.

[9] A. H. Karp and H. P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, May 1990.

[10] A. Grama, A. Gupta, and V. Kumar. Isoefficiency function: A scalability metric for parallel algorithms and architectures. Technical Report TR 93–24, University of Minnesota, Department of Computer Science, 1993.

[11] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical report, Rept 95-064, Department of Computer Science, University of Minnesota, 1995.

[12] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA highly scalable server. *ACM*, 23:241–251, 1997.

[13] K. H. Law. A parallel finite element solution method. *Computers and Structures*, 23(6):845–858, 1988.

[14] C. Farhat and L. Crivelli. A general approach to nonlinear FE computations on shared memory multiprocessors. *Computer Methd. in Appl. Mech. Eng.*, 72:153–171, 1989.

[15] D. C. Hodgson and P. K. Jimack. Efficient mesh partitioning for parallel elliptic differential equation solvers. *Computing Systems in Engineering*, 6(1):1–12, 1995.

[16] A. Gupta, V. Kumar, and A. Samesh. Performance and scalability of preconditioned conjugate gradient methods on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 6(5):455–469, 1995.

[17] W. Gropp. *Parallel Computing and Domain Decomposition*, chapter 29. 5th Int. Symposium on DDM for partial Differential equations, SIAM, 1991.

[18] Y. A. Kuznetsov. *Overlapping Domain Decomposition Methods for FE-Problems with Elliptic Singular Perturbed Operators*, chapter 21. SIAM, 1991.

[19] J. H. Saltz, V. K. Naik, and D. M. Nicol. Reduction of the effects of the communication delays in scientific algorithms on message passing MIMD architectures. *SIAM J. Sci. Stat. Comput.*, 8(1):118–134, Januray 1987.

[20] G. Yagawa, A. Yoshioka, S. Yoshimura, and N. Soneda. A parallel finite element method with a supercomputer network. *Computers and Structures*, 47(3):407–418, 1993.

[21] Y. Escaig, G. Touzot, and Michel Vayssade. Parallelization of a multilevel domain decomposition method. *Computing Systems in Engineering*, 5(3):253–263, 1994.

[22] H. F. Jordan. Interpreting parallel processor performance measurements. *SIAM J. Sci. Stat. Comput.*, 8(2):220–226, March 1987.

[23] A. Skjellum, E. Lusk, and W. Gropp. Early applications in the message-passing interface (MPI). *The Int. J. Sup. Appl.*, 9(2):79–94, 1995.

[24] B. N. Maker, Q. Jiangning, and D. T. Nguyen. Performance of NIKE3D with PVSOLVE on vector and parallel computers. *Computing Systems in Engn.*, 5(4–6):963–368, 1994.

[25] P. Henriksen and R. Keunings. Parallel computation of the flow of integral viscoelastic fluids on a heterogeneous network of workstations. *Int. J. Numer. Methods Fluids*, 18:1167–1183, 1994.

[26] M. E. M. El-Sayed and C. K. Hsiung. Parallel finite element computation with separate substructures. *Computers and Structures*, 36(2):261–265, 1990.

[27] A. I. Khan and B. H. V. Topping. Subdomain generation for parallel finite element analysis. *Computing Systems in Engineering*, 4(4–6):473–488, 1993.

[28] D. E. Keyes and W. Gropp. A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation. *SIAM J. Sci. Stat. Comput.*, 8(2):166–202, 1987.

[29] K. H. Ahn and D. A. Hopkins. A generalized domain decomposition technique for mixed iterative finite element formulation. *Computing Systems in Engineering*, 1994.

[30] V. Kumar and A. Gupta. Analyzing scalability of parallel algorithms and architectures. Technical Report TR 91–18, University of Minnesota, Department of Computer Science, 1993.